

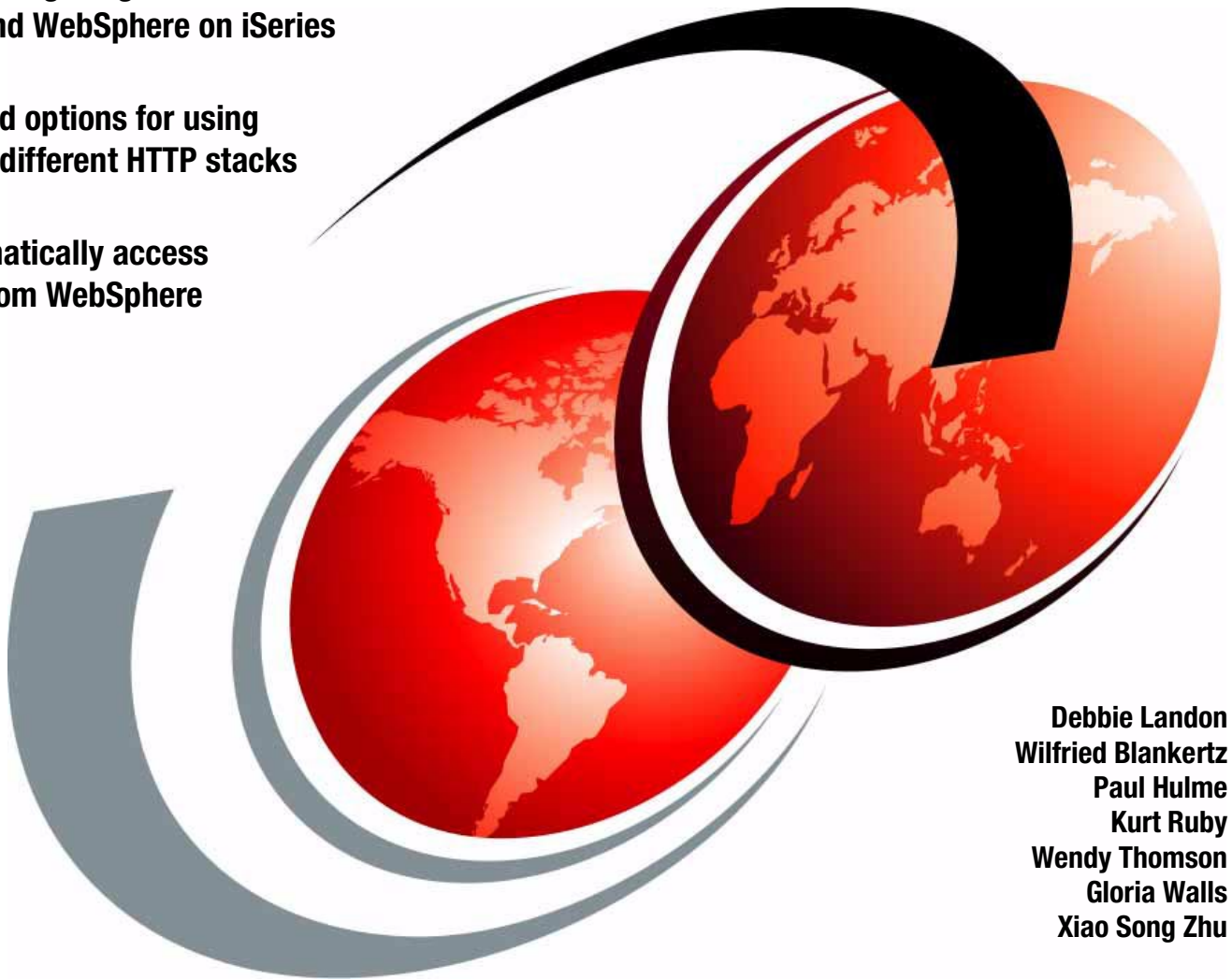
# Domino and WebSphere Integration

on the IBM  iSeries Server

Configure single sign-on between  
Domino and WebSphere on iSeries

Understand options for using  
LDAP and different HTTP stacks

Programmatically access  
Domino from WebSphere



Debbie Landon  
Wilfried Blankertz  
Paul Hulme  
Kurt Ruby  
Wendy Thomson  
Gloria Walls  
Xiao Song Zhu





International Technical Support Organization

**Domino and WebSphere Integration  
on the IBM @server iSeries Server**

March 2002

**Take Note!** Before using this information and the product it supports, be sure to read the general information in “Special notices” on page 301.

#### **First Edition (March 2002)**

This edition applies to Lotus Domino for AS/400 R5.0.6a or later and IBM WebSphere Application Server Advanced Edition Version 3.5.2 or later for use with OS/400 V4R5 and later.

Comments may be addressed to:  
IBM Corporation, International Technical Support Organization  
Dept. JLU Building 107-2  
3605 Highway 52N  
Rochester, Minnesota 55901-7829

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

**© Copyright International Business Machines Corporation 2002. All rights reserved.**

Note to U.S Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Preface</b> .....	vii
The team that wrote this redbook. ....	vii
Notice .....	ix
IBM trademarks .....	ix
Comments welcome. ....	ix
 <b>Chapter 1. Overview</b> .....	1
1.1 Collaborative commerce .....	2
1.1.1 What is WebSphere? .....	2
1.1.2 Comparing Domino and WebSphere technologies .....	4
1.2 Points of integration .....	6
1.3 Supported features .....	7
1.4 How this book is organized .....	9
 <b>Chapter 2. Prerequisites</b> .....	11
2.1 Workstation requirements .....	12
2.2 iSeries requirements .....	12
2.3 Creating a WebSphere Application Server instance .....	13
2.4 Using the code examples .....	14
 <b>Part 1. Administration topics</b> .....	15
 <b>Chapter 3. HTTP options</b> .....	17
3.1 Architecture overview .....	18
3.2 Domino HTTP server support .....	19
3.2.1 Configuring Domino HTTP server for use by WebSphere .....	20
3.2.2 Using Domino HTTP server to call a WebSphere servlet .....	22
3.2.3 Using Domino HTTP server to manage multiple Web sites .....	25
3.3 iSeries HTTP server support .....	34
3.3.1 Configuring Domino to use iSeries HTTP server .....	35
3.3.2 Using iSeries HTTP server to manage multiple Web sites .....	44
3.3.3 Switching from iSeries HTTP server to Domino HTTP server .....	52
3.4 Running Domino and iSeries HTTP servers together .....	52
3.4.1 Configuring Domino to use a different port for HTTP .....	52
3.4.2 Defining a virtual host for WebSphere to recognize Domino calls .....	53
 <b>Chapter 4. Directory sharing using LDAP</b> .....	57
4.1 What is LDAP? .....	58
4.1.1 Options for directory sharing using LDAP on the iSeries server .....	59
4.2 Using OS/400 LDAP .....	60
4.2.1 Configuring OS/400 LDAP .....	61
4.2.2 Starting and stopping the OS/400 LDAP server .....	67
4.2.3 Publishing OS/400 SDD entries to the LDAP directory .....	68
4.2.4 Verifying the connection to the OS/400 LDAP server .....	74
4.2.5 Managing directory entries .....	75
4.2.6 Using LDAP for authentication .....	78
4.2.7 Saving the LDAP directory and problem determination .....	78
4.3 Configuring WebSphere and Domino to use OS/400 LDAP .....	80
4.3.1 Authentication concepts of the WebSphere Application Server .....	80

4.3.2	Enabling global security in the WebSphere Application Server . . . . .	81
4.3.3	Creating a Directory Assistance Domino database. . . . .	88
4.3.4	Configuring Domino to use OS/400 LDAP . . . . .	91
4.4	Using Domino LDAP . . . . .	94
4.4.1	Configuring the Domino Directory for LDAP access. . . . .	95
4.4.2	Starting and stopping the Domino LDAP task. . . . .	97
4.4.3	Verifying the connection to the Domino LDAP server. . . . .	99
4.5	Configuring WebSphere to use Domino LDAP . . . . .	101
4.5.1	Authentication concepts of the WebSphere Application Server . . . . .	102
4.5.2	Enabling global security in the WebSphere Application Server . . . . .	103
4.6	Securing WebSphere resources . . . . .	109
4.6.1	Creating an enterprise application . . . . .	109
4.6.2	Configuring application security . . . . .	112
4.6.3	Configuring resource security . . . . .	115
4.6.4	Configuring security permissions . . . . .	118
4.6.5	Starting the enterprise application . . . . .	121
<b>Chapter 5.</b>	<b>Single sign-on . . . . .</b>	<b>123</b>
5.1	What is single sign-on? . . . . .	124
5.2	SSO prerequisites . . . . .	124
5.3	Which LDAP server should be used? . . . . .	125
5.4	Configuring SSO for WebSphere . . . . .	125
5.4.1	Updating the global security settings for WebSphere . . . . .	125
5.4.2	Exporting the LTPA keys . . . . .	129
5.5	Configuring SSO for Domino. . . . .	130
5.5.1	Creating the Domino Web SSO Configuration document. . . . .	130
5.5.2	Configuring the Domino server document. . . . .	133
5.6	Verifying single sign-on between WebSphere and Domino . . . . .	136
5.6.1	Verifying Domino and WebSphere security without SSO enabled . . . . .	137
5.6.2	Verifying Domino and WebSphere security with SSO enabled. . . . .	139
5.7	Problem determination . . . . .	145
<b>Part 2.</b>	<b>Programming topics . . . . .</b>	<b>149</b>
<b>Chapter 6.</b>	<b>Using WebSphere and the Domino object models . . . . .</b>	<b>151</b>
6.1	WebSphere topology. . . . .	152
6.2	Using Java to access Domino from WebSphere. . . . .	153
6.2.1	Local Domino objects . . . . .	154
6.2.2	Remote Domino objects . . . . .	154
6.3	Configuring a WebSphere Application Server. . . . .	156
6.3.1	Configuring access to Domino using remote classes . . . . .	156
6.3.2	Configuring access to Domino using the local classes . . . . .	169
6.3.3	Performance of local versus remote access . . . . .	182
6.4	Application development with single sign-on . . . . .	182
<b>Chapter 7.</b>	<b>Accessing Domino from WebSphere . . . . .</b>	<b>183</b>
7.1	Development environment . . . . .	184
7.1.1	Application overview . . . . .	184
7.1.2	Setup of VisualAge for Java . . . . .	186
7.1.3	Creating the Domino database . . . . .	191
7.2	Invoking servlets . . . . .	197
7.2.1	Servlet access to Domino . . . . .	197
7.3	Using JavaServer Pages. . . . .	211
7.3.1	Extending the Snow Dome application using JSPs . . . . .	211

7.4 Using Enterprise JavaBeans . . . . .	217
7.4.1 Using Enterprise JavaBeans to access Domino . . . . .	218
7.4.2 Creating access beans and deployment code . . . . .	234
7.5 Application deployment . . . . .	236
7.5.1 Using the EJB test client . . . . .	236
7.5.2 Using the WebSphere Test Environment . . . . .	241
7.5.3 Deployment to WebSphere Application Server . . . . .	242
7.6 Using Enterprise JavaBeans from a Domino agent . . . . .	260
<b>Part 3. Appendixes . . . . .</b>	<b>263</b>
<b>Appendix A. Code examples . . . . .</b>	<b>265</b>
HTML and JSP files . . . . .	266
SDIndex.html . . . . .	266
Registration.html . . . . .	266
RegWJSP.html . . . . .	267
Regejb.html . . . . .	268
SDSearch.jsp . . . . .	269
SDattendee.jsp . . . . .	270
SDsuccessUpdate.jsp . . . . .	271
CustDisplay.jsp . . . . .	272
ReturnMessage.jsp . . . . .	273
GetCustomer.html . . . . .	273
Java servlets . . . . .	274
SimpleRemote.java . . . . .	274
SimpleLocal.java . . . . .	276
Register.java . . . . .	278
RegisterWJSP.java . . . . .	280
Controller.jsp . . . . .	283
JavaBean . . . . .	285
CustInfoBean.java . . . . .	285
Enterprise JavaBean . . . . .	286
RegDocBean.java . . . . .	286
VisualAge for Java configuration file . . . . .	293
default_app.webapp . . . . .	293
<b>Appendix B. Additional material . . . . .</b>	<b>295</b>
Locating the Web material . . . . .	295
Using the Web material . . . . .	295
Installing the Web material . . . . .	296
<b>Special notices . . . . .</b>	<b>301</b>
<b>Related publications . . . . .</b>	<b>303</b>
IBM Redbooks . . . . .	303
Other resources . . . . .	303
Referenced Web sites . . . . .	303
How to get IBM Redbooks . . . . .	304
IBM Redbooks collections . . . . .	304
<b>Glossary . . . . .</b>	<b>305</b>
<b>Index . . . . .</b>	<b>309</b>





# Preface

IBM WebSphere and Lotus Domino are both strategic products for e-business and business-to-business or B2B solutions. Domino and its related products deliver collaborative commerce services, while WebSphere allows for Java-based Web applications with a strong transactional emphasis.

This IBM Redbook documents the setup and configuration of an integrated environment with WebSphere and Domino on the IBM @server iSeries server. Part 1 of this redbook focuses on administration topics, such as single sign-on (SSO) and the use of a common LDAP directory for authentication, as well as options for different HTTP stacks.

Part 2 of this redbook focuses on investigating the application development topic of how WebSphere can access Domino applications. It shows and discusses code snippets from example programs. The example programs are available for download. To understand this code better, download the files, as explained in Appendix B, "Additional material" on page 295, and use them as a reference.

The examples in this redbook were developed using VisualAge for Java Enterprise Edition 3.5.3 and tested using WebSphere Application Server 3.5.3. To use this book effectively, you should be familiar with the Java programming language and the concepts of Enterprise JavaBeans.

**Note:** This redbook reflects the IBM @server iSeries server name. Throughout this redbook, we use the shortened version "iSeries" to refer to both AS/400e and iSeries servers. Although the examples in this redbook were conducted using the AS/400e running V4R5, they should work the same way on iSeries servers running V4R5.

## The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Rochester Center.

**Debbie Landon** is an IT iSeries Specialist at the International Technical Support Organization (ITSO), Rochester Center, focusing on Domino for iSeries. She writes extensively and teaches IBM classes worldwide on Domino for iSeries. Before joining the ITSO in November of 2000, she was a member of the PartnerWorld for Developers, iSeries team supporting business partners in the area of Domino for iSeries. She was a member of the original team that created the highly successful Domino Days event, which has since been replicated worldwide.

**Wilfried Blankertz** is a Technical Support Specialist for iSeries in IBM EMEA Region Central located in Frankfurt, Germany. From 1995 to 1998 he was assigned to the International Technical Support Organization, Rochester Center and wrote extensively and taught IBM classes worldwide on all areas of AS/400 Groupware solutions and Systems Management. Before joining the ITSO, he worked as a systems engineer in IBM Germany supporting customers with the AS/400 system and its predecessor systems (IBM System /3, /32, /34, /36, and /38) for over 25 years. While he still focuses on iSeries technical support, he is also a Certified Lotus Professional for Domino R5 Application Development and Domino Administration.

**Paul Hulme** is an e-business specialist working with Internet Changes located in Melbourne, Australia. He specializes in application development, consulting services, and solutions architecture. His areas of expertise include distributed application development, IBM WebSphere, VisualAge for Java, and Lotus Domino. He has experience in working with all these products on a variety of platforms. His e-mail address is paul@itch.com.au.

**Kurt Ruby** is an Advisory Programmer in the iSeries Custom Technology Center at IBM in Rochester, Minnesota. He has 18 years of experience with IBM and for the past four years has been working on developing applications with Lotus Domino. He holds a bachelor's degree in Mathematics from Iowa State University. His areas of expertise include iSeries Domino database integration, Domino performance, and Domino Java interface technologies. He has written exclusively on Lotus Domino for AS/400.

**Wendy Thomson** is a contractor with IBM Australia, working both in Australia and in the U.S. since 1992. Over this period she has been contracted to IBM Learning Services, ITSO and PartnerWorld for Developers, iSeries. She has worked for the ITSO Rochester Center on redbook residencies and writing education course material and presentations. She has written Internet-based education courses for the PartnerWorld for Developers, iSeries team. And, as an Instructor for IBM Learning Services, she teaches AS/400 and iSeries classes to customers, business partners and IBM internal staff.

**Gloria Walls** is a Senior Consultant with Technology Genesys, Inc., located in Austin, Texas. She is a Principal Certified Lotus Instructor, R5 Principal Certified Lotus Professional in Application Development, R5 Certified Lotus Professional in System Administration, IBM Certified Specialist in IBM WebSphere Application Server Standard Edition, V 3.5 and Sun Certified Programmer for the Java 2 Platform. She specializes in e-business application development and training. Gloria has extensive experience with Lotus Domino, and her job responsibilities have included application development, technical support, consulting and conducting IBM and Lotus Authorized training courses for developers and administrators. Her e-mail address is gloria@techgenesys.com.

**Xiao Song Zhu** is a Technical Support Specialist for iSeries in IBM China. She is responsible for technical support on AS/400 in China, including system management and e-business solutions, especially Domino for AS/400. Her various support roles have included strategy, architecture, project design and implementation, system performance tuning and training.

Thanks to the following people for their invaluable contributions to this project:

Marla Berg  
Pat Fleming  
Dave Herbeck  
Don Morrison  
Sanjay Patel  
iSeries development team, Rochester

Lisa Woody  
IBM Lotus Integration Center (ILIC), Roanoke, Texas


Bob Matta  
International Technical Support Organization, Rochester Center


## Notice

This publication is intended to help system administrators, programmers, and technical specialists to understand how to integrate Domino and WebSphere on the iSeries server from both a system administration and a programmatic point of view. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM iSeries software, including OS/400, Domino, or WebSphere. See the PUBLICATIONS section of the IBM Programming Announcement for these products for more information about what publications are considered to be product documentation.

## IBM trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

e (logo)®   
AIX®  
AS/400®  
AS/400e™  
DB2®  
Everyplace™  
IBM®  
IBM.COM™  
iSeries™  
OS/390®  
OS/400®  
PartnerWorld®

Redbooks Logo   
Redbooks™  
S/390®  
SecureWay®  
SP™  
SP1®  
VisualAge®  
WebSphere®  
Lotus®  
Lotus Notes®  
Notes®  
Domino™  
QuickPlace™

## Comments welcome

Your comments are important to us!

We want our IBM Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:  
[ibm.com/redbooks](http://ibm.com/redbooks)
- ▶ Send your comments in an Internet note to:  
[redbook@us.ibm.com](mailto:redbook@us.ibm.com)
- ▶ Mail your comments to the address on page ii.





# Overview

In this first chapter, we provide an overview of Domino and WebSphere integration on the iSeries server. We briefly describe what WebSphere is and do a quick comparison between the Domino and WebSphere technologies. Finally, we describe the various points of integration between Domino and WebSphere specifically on the iSeries server.

## 1.1 Collaborative commerce

Both Domino and WebSphere are an important part of IBM's e-business strategy. Domino and WebSphere are middleware – the arena where integration happens. Together they provide the platform for collaborative commerce on the iSeries server.

Commerce has grown up from being primarily about content to enabling transactions (the capabilities to submit purchase orders). More recently, however, commerce has been about collaborative relationships. This has come about as a result of several factors. First, commerce between companies or business-to-business (B2B) commerce has become more prevalent in the industry. B2B commerce requires a much greater need for relationships and collaborative technologies to facilitate purchasing of large, complex transactions between teams of people. Second, sites with transactional capabilities are realizing that they could increase the amount and frequency of purchasers by assisting transactions. Third, as commerce sites grow up, they have realized that there is much more about the buying and selling process than the transaction.

So how do customers decide which product is right for them? The answer is it depends on what you're trying to build. If you're building Web applications that focus on collaboration, Domino is the right choice. Domino excels when an application requires sophisticated collaborative capabilities, such as workflow, distributed content authoring, or deals with a lot of unstructured data or knowledge. If you're building a Web site that needs to handle a lot of transactions or requires the use of Enterprise Java components, then WebSphere is the right choice. WebSphere excels when an application requires industrial-strength transaction management, massive scalability, or where business logic is wholly encapsulated in distributed components such as servlets or Enterprise JavaBeans (EJBs). If you need to build a site that needs both of these capabilities, then Domino and WebSphere can work together over the same or separate HTTP stacks to deliver the best of both worlds.

For example, one customer has built an application that uses a Domino workflow infrastructure to implement content management for their Web site catalog. Domino-based workflow and collaboration software is used to manage the content generation and content approval process for catalog entries and prices. The highly scalable transaction services of WebSphere are used in tandem to implement the actual catalog site. Domino messaging is used to send confirmation information to the customer and to maintain the dialog with the customer.

### 1.1.1 What is WebSphere?

So what exactly is WebSphere? We will give a brief description here. For an in-depth explanation and description, refer to the redbook *WebSphere V3.5 Handbook*, SG24-6161.

WebSphere is actually an IBM product brand name encompassing a family of products including:

- ▶ WebSphere Application Server
- ▶ WebSphere Studio
- ▶ WebSphere Performance Pack
- ▶ WebSphere Performance Pack Cache Manager
- ▶ WebSphere Transcoding Publisher
- ▶ WebSphere Commerce Suite
- ▶ WebSphere Catalog Architect
- ▶ WebSphere Solution Components
- ▶ WebSphere Everyplace Suite

The major product is WebSphere Application Server, a Web application server that delivers Java-based applications built with combinations of three component types: servlet, JavaServer Pages (JSP), and Enterprise JavaBean (EJB). These components are not unique to WebSphere; all Web application servers use them.

WebSphere runs independently of Domino. It can plug into just about any commercially available HTTP server, such as the IBM HTTP server, Microsoft's Internet Information Server (IIS), Apache, and Lotus Domino R5. WebSphere is a server process, not something you would run on a client computer. It handles certain kinds of URLs (passed over to the WebSphere task by the HTTP task) to process invocations of servlets or JavaServer Pages (JSPs). The Advanced and Enterprise Editions of WebSphere also implement a technology called Enterprise JavaBeans or EJBs. Unlike servlets or JSPs, EJBs are not invoked via the HTTP through the Web server, but are started using another network protocol called Internet Inter-ORB Protocol (IIOP). EJBs are typically called from servlets.

WebSphere comes in several flavors: Standard, Advanced, and Enterprise Editions. There are many differences between the different editions. You can find detailed descriptions on the IBM Web site at: <http://www.ibm.com/software/webservers/appserv>

The WebSphere environment is shown in Figure 1-1.

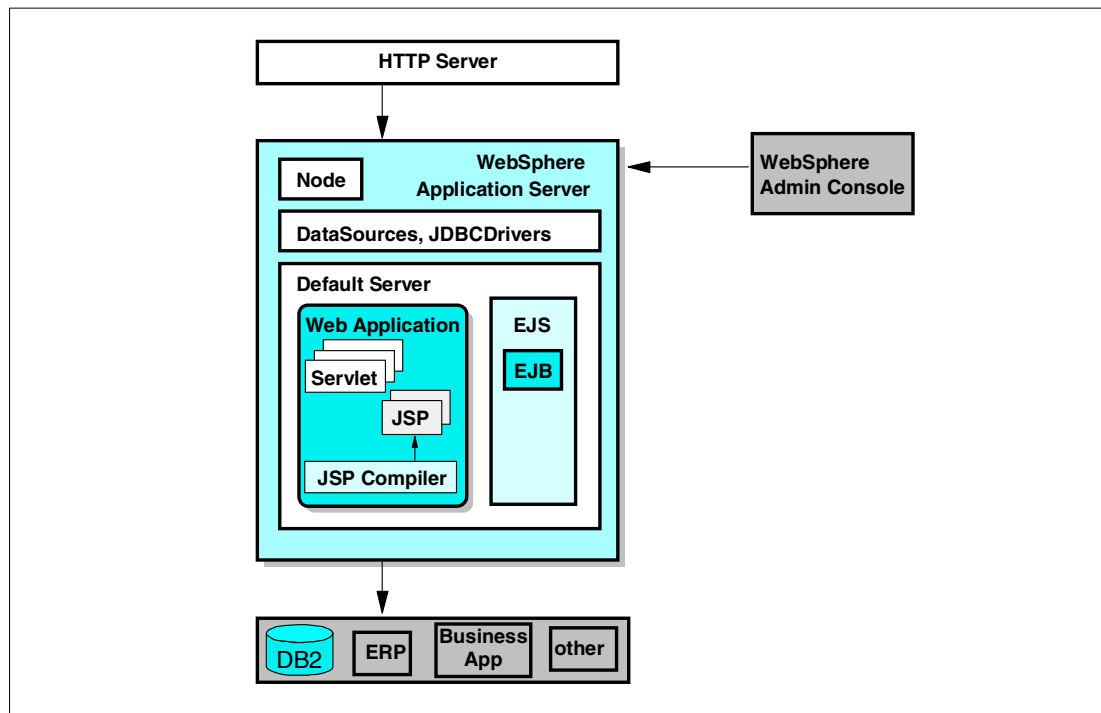


Figure 1-1 WebSphere environment

## WebSphere programmability

All versions of WebSphere are programmed with Java, although you have a few different options for the kind of Java programming you want to do:

### ► Servlets

A Java servlet is analogous to a Domino agent. Servlets came about from the earlier CGI (Common Gateway Interface) programming model, where an HTTP server would, in response to a particular kind of URL request, launch an executable program (C, C++, Perl, or some other programming language) on the server to handle the request and return an

HTML result. Java servlets are much more efficient because they do not require a process startup and shutdown on each request. The Java virtual machine (JVM) runs all the time in the Servlet Engine's (WebSphere's) process space, and the Java code for the servlet can also be cached in memory.

The servlet programming model also provides special CGI-BIN tools simplifying the programmer's job. There are other classes that provide for persistence and pooling of resources. These combine to make powerful and scalable Web applications.

► **JavaServer Pages (JSPs)**

JSPs are very much like Microsoft's Active Server Pages (ASPs), where Web pages can be mixed with static HTML content with programmed dynamic content (HTML) using VBScript, Java or JavaScript. WebSphere JSPs, currently supports only Java for dynamic content. When a JSP URL is invoked, WebSphere runs the page as if it were a servlet, combining the static HTML in the page with the HTML created dynamically by the embedded Java code in the page.

► **Enterprise JavaBeans (EJBs)**

EJBs are a component programming model for dealing in a nice way with back-end transactional data systems, typically relational DBMSs or ERPs. You write relatively simple Java classes that conform to certain conventions and the WebSphere EJB "container" handles much of the details of database access, two-phase commit coordination, remote communications, and database access. WebSphere Advanced Edition contains support for EJBs working with single-transacted data sources, while the Enterprise Edition supports transaction commit and rollback across multiple data sources.

For details on programming servlets and JSPs, refer to the redbook *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, SG24-5755.

## 1.1.2 Comparing Domino and WebSphere technologies

Both WebSphere and Domino are extremely popular tools for building Web-based applications. Figure 1-2 shows that the typical types of applications that people tend to build with Domino or WebSphere are very different. For Domino it's the people-centric, document-based applications such as TeamRooms, Knowledge Bases and intranets. Domino, in a nutshell, is great when people need to deal with documents, or especially when they have to share them, or move them around in a workflow. For WebSphere it's the systems-centric, data-based applications such as storefronts, Supply Chain Integration and extranets. WebSphere, in a nutshell, is great when you want to extend back-end systems to the Web, as an extranet or wrapped into an e-commerce site or B2B exchange.

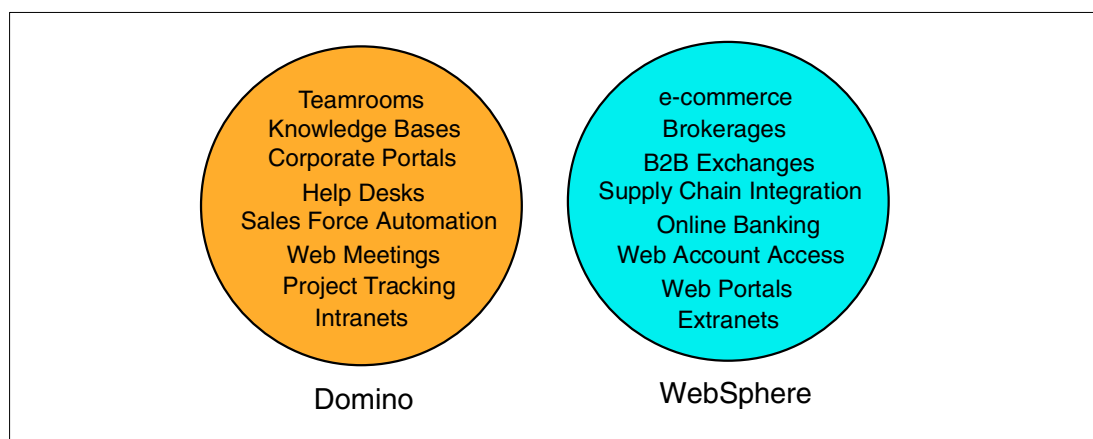


Figure 1-2 Typical Domino and WebSphere applications



The reasons that people use Domino and WebSphere for different things is that they have different underlying capabilities, as shown in Figure 1-3. Domino has its world-class routing and document database, its unrivaled replication and agents technology. There are Domino features around awareness and application sharing available in the Sametime and QuickPlace products. Then there are a whole bunch of "plumbing" functions that Domino needs to have as an application server -- page rendering, enterprise integration, a development model, Internet standards support, etc. WebSphere has rich support for transactions, a library of business objects, shopping cart services and payment settlement. And, it too has all those other "plumbing" functions you need when you're an application server.

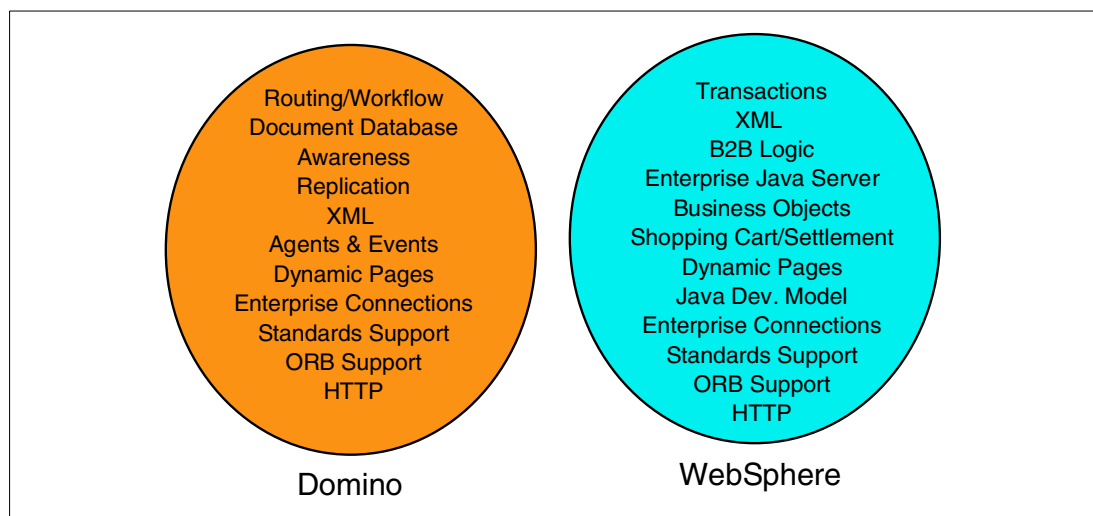


Figure 1-3 Domino and WebSphere functions

When you integrate Domino and WebSphere together, you are taking all those common "plumbing" functions and merging them together as shown in Figure 1-4, so that whether you buy Domino or WebSphere, those things will always be the same. That way, as you put together your application server infrastructure, you know that you'll be able to easily add Domino products to your WebSphere platform, and vice versa. The result is very simple but very powerful and dynamic. As a result of the integration, it will be easy for you to, for example, add Teamrooms to your B2B Exchange, add awareness to your e-commerce site, or add knowledge base access to your Web account access applications.

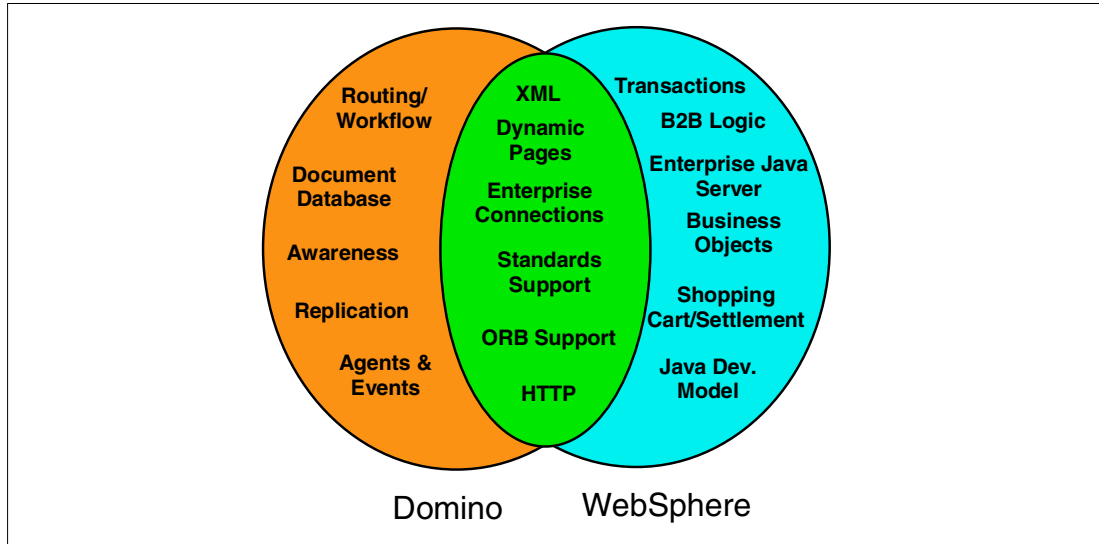


Figure 1-4 Domino and WebSphere integrated

## 1.2 Points of integration

Because WebSphere programs are written in Java, it is easy to write servlets, JSPs, or EJBs that make use of the Domino Java API. In addition there are several other points of integration between Domino and WebSphere on the iSeries server:

- ▶ HTTP services

Domino comes with its own HTTP server task. WebSphere can use the Domino HTTP server or it can use the OS/400 HTTP server. Similarly Domino can be configured to use the OS/400 HTTP server instead of its own.

- ▶ Directory services

Having a Lightweight Directory Access Protocol (LDAP)-compatible directory available somewhere is crucial for implementing any kind of security with WebSphere. LDAP is a network protocol that lets any application access directory information and perform authentication services, such as validating user names and passwords. The application has to validate the user's identity and implement some kind of access control based on that identity. Domino has its own directory as part of the product; WebSphere has no built-in directory at all. Also, when implementing the single sign-on function between WebSphere and Domino, an LDAP directory is mandatory.

When integrating Domino and WebSphere on the iSeries, you basically have two choices for an LDAP directory: either to use the LDAP directory support in Domino or use the OS/400 directory services support for LDAP.

- ▶ Single sign-on

A single sign-on capability spans these two products. This allows an application to use services from either Domino or WebSphere, while only requiring a single user login. That means you'll be able to initiate a transaction from WebSphere and have the back-end execute some aspects of it on Domino under the same login credentials. The opposite will also be possible; you'll be able to start a transaction on Domino and have some aspects of it execute in WebSphere, again under the same identity. The goal is to provide a seamless flow of information across these products so that the boundary defined by which bits get shipped in which box becomes unimportant for the purposes of building applications.

## 1.3 Supported features

Table 1-1 shows when the various Domino and WebSphere integration features are supported with the various releases of Domino for iSeries and WebSphere on the iSeries server. The following paragraphs provide brief explanations of the features listed in Table 1-1.

### ***Domino plug-in for OS/400 HTTP server***

The Domino plug-in for OS/400 HTTP server allows you to perform iSeries Web serving of both Domino and non-Domino content using a single Web server. With this support, Domino can be set up to use the iSeries HTTP server instead of its own internal HTTP server. For details on how to configure this support, refer to Section 3.3, “iSeries HTTP server support” on page 34.

### ***Running WebSphere with Domino HTTP stack***

The HTTP server of Domino can be used as the Web server for the WebSphere Application Server. The Domino HTTP server can be used for serving both Domino and WebSphere Web content. For details on how to configure this support, refer to Section 3.2, “Domino HTTP server support” on page 19.

### ***Single sign-on across multiple Domino Web servers***

A Web user can now log on once to a Domino server, then access any other Domino server in the same domain without logging on again. This is accomplished with a new “multi-server” option in the Domino server document for session-based authentication, along with a new Configuration document in the Domino Directory called the Web SSO document. Note that all Domino servers participating in multi-server session authentication must be at the Domino R5.0.5 level or above, and users’ Web browsers must have cookies enabled.

### ***Single sign-on between Domino and WebSphere***

A Web user can now log on once to a Domino server, then access any WebSphere Application Server 3.5.1 or higher without logging on again. In Domino this is accomplished by setting up the new “multi-server” option for session-based authentication in the Domino server document and importing the secret key from a WebSphere server rather than generating it in Domino. The WebSphere server(s) must be configured in secure mode and use WebSphere’s LTPA (Lightweight Third-Party Authentication) protocols. For details on configuring single sign-on between Domino and WebSphere, refer to Chapter 5, “Single sign-on” on page 123.

The WebSphere server(s) may use the Domino Directory as the repository for user definitions and passwords or certificates, via Domino’s LDAP service. Note that all servers participating in a single sign-on domain must be at the Domino 5.0.5 or above and WebSphere Application Server 3.5.1 or higher.

### ***Domino and WebSphere sharing an LDAP directory for authentication***

WebSphere and Domino have the capability to share a common directory for users and groups. This common directory can be used to authenticate users and determine their membership in a group. There are two basic approaches to establishing a common directory shared by Domino and WebSphere on the iSeries. One is to use the Domino Directory as the common directory for the WebSphere and Domino server. The other is to use another LDAP-compatible directory service as the common directory. For this approach, the directory service choices are preferably those explicitly supported by WebSphere and Domino. For example on iSeries, this would be the IBM SecureWay Directory for OS/400 product. For details on configuring a common LDAP directory for Domino and WebSphere, refer to Chapter 4, “Directory sharing using LDAP” on page 57.

### **Java API options for authenticated invocation**

New options in the Domino APIs for Java allow sessions between Domino servers, or calls between Domino and WebSphere 3.5 Java code, to carry a user's authenticated identity. The servers must be set up to utilize the single sign-on features introduced in Domino R5.0.5 and WebSphere 3.5.1.

### **WebSphere accessing Domino objects with Java locally or remotely**

The WebSphere servlets, JSPs and EJBs that you develop may have a need to access information or functions that are available in a Domino server environment. The Domino Java API provides two possible separate object models that would allow you to access Domino functionality and services. One choice is to use the Local Domino Object Model and the other is to use the Remote Domino Object Model. For details on using these two Domino Java API object models, refer to Section 6.2, "Using Java to access Domino from WebSphere" on page 153.

### **WebSphere accessing Domino objects via JDBC**

The capability to access Domino objects through JDBC currently only works on Windows NT. Currently the iSeries server does not support this function.

Table 1-1 What features are supported in which releases of Domino for iSeries and WebSphere on iSeries

Feature	Domino for iSeries			WebSphere on iSeries			
	5.0.4	5.0.5	5.0.6a <sup>+</sup>	3.0 <sup>+</sup>	3.5	3.5.1	3.5.2 <sup>+</sup>
Domino plug-in for OS/400 HTTP Server (Domino using OS/400 HTTP stack)	5.0.4a	X	X				
Running WebSphere with Domino HTTP stack		X	X			X	X
Single sign-on across multiple Domino Web servers		X	X				
Single sign-on between Domino and WebSphere		X <sup>1</sup>	X			X	X
Domino and WebSphere sharing an LDAP directory for authentication (Domino or OS/400 LDAP)	X	X	X	X	X	X	X
Java API options for authenticated invocation		X	X		X	X	X
WebSphere accessing Domino objects with Java locally (uses NOTES.jar)	X	X	X	3.02 <sup>2</sup>	X <sup>2</sup>	X <sup>2</sup>	X <sup>2</sup>
WebSphere accessing Domino objects with Java remotely (uses NCSOW.jar)	X	X	X	X	X	X	X
WebSphere accessing Domino objects via JDBC <sup>3</sup>	Not currently supported on the iSeries server			Not currently supported on the iSeries server			

The following numbers refer to the superscripts in Table 1-1:

1. Not supported on the iSeries server.
2. Local access via the NOTES.jar to Java APIs for Domino is not recommended from WebSphere. It is recommended that you use remote access via the NCSOW.jar and the IIOP task in Domino.
3. The JDBC driver does not currently support accessing Domino objects on the iSeries server.

## 1.4 How this book is organized

This redbook has two parts. Part 1 focuses on administration and covers the following topics:

- ▶ HTTP options
- ▶ Directory sharing using LDAP
- ▶ Single sign-on

Part 2 of this redbook focuses on programming and covers the following topics:

- ▶ Using WebSphere and the Domino object models
- ▶ Accessing Domino from WebSphere





## Prerequisites

This chapter covers what you need to install and configure on your iSeries server and PC client in order to use the examples covered in this book.

## 2.1 Workstation requirements

This section describes the requirements for the administrative workstation used to install, configure, and manage an integrated Domino and WebSphere environment on the iSeries server. The required software is:

- ▶ Web browser; in our examples we used Netscape 4.75
- ▶ Lotus Domino Administrator client R5.08
- ▶ WebSphere Administrative Console

The WebSphere Administrative Console provides a Java graphical user interface (GUI). Since the iSeries server does not support native GUI devices, it cannot run directly on the iSeries server. You must use a Windows or AIX workstation to run the WebSphere Administrative Console locally.

For details on installing and configuring the WebSphere Administrative Console to manage a WebSphere Application Server on the iSeries server, refer to the redbook *Building iSeries Applications for WebSphere Advanced Edition 3.5*, SG24-5691.

All the code examples used in Part 2, “Programming topics” on page 149, of this redbook were developed and tested with the following software releases:

- ▶ VisualAge for Java 3.5.3 Enterprise Edition
- ▶ An HTML editor such as WebSphere Studio 3.5
- ▶ Lotus Domino Designer R5.08

**Note:** The installation and configuration of each of the products listed above is not within the scope of this book. For detailed information, refer to the respective product documentation.

## 2.2 iSeries requirements

This section describes the requirements for the iSeries server used to install, configure, and manage an integrated Domino and WebSphere environment.

The versions of software used in this redbook include:

- ▶ Lotus Domino for iSeries R5.0.8
- ▶ IBM WebSphere Advanced Edition V3.5.3

For an excellent source of information on the details of installing and configuring WebSphere on the iSeries server, refer to the redbook *Building iSeries Applications for WebSphere Advanced Edition 3.5*, SG24-5691.

- ▶ OS/400 V4R5
- ▶ Client Access Express

**Attention:** As is always the case, newer versions of these software products are continually being made available.

This redbook assumes that the iSeries server already has TCP/IP configured, and the administrative PC client workstation being used can ping the iSeries server. You must also have QSECOFR authority access in order to complete some steps in this book.



## 2.3 Creating a WebSphere Application Server instance

The iSeries version of WebSphere Application Server supports multiple instances of WebSphere Application Server. The advantage of creating multiple instances is the ability to have concurrent but completely independent versions of the instances. Each server instance can read from its own set of properties files, create its own set of log and trace files, work within its own security model, and can invoke its own Administration Manager interface without affecting any other server instances that might be running on the iSeries server at the same time.

The ability to have multiple instances allows you to keep application developers independent from each other. For example, you could use one instance for development, one for testing, and one for serving the Web site. Each of these instances will be independent. Changes made to one environment will not affect other environments. You will also find multiple instances of WebSphere Application Server useful if you are training several people on WebSphere or running a workshop. Each person can have his or her own instance running on the same iSeries server. Each individual will be able to make configuration changes, as well as start and stop his own WebSphere Application Server instance without impacting others.

To follow the examples throughout this book, you will need access to a WebSphere Application Server instance. We will be changing the configuration of this instance frequently and it may be more desirable for you to create a new instance for the purpose of following these examples. To create a new WebSphere Application Server instance, perform the following steps:

1. Creating a WebSphere Application Server instance is easy. There is a Java program that comes with WebSphere that does everything for you. Enter the OS/400 Qshell Interpreter and run the script that creates all new server directories and sets up the correct authorities. From a 5250 emulation session, start the Qshell Interpreter by entering either the STRQSH or QSH command on the OS/400 command line.
2. In the QShell environment, enter the following command (in a single line) to create your instance of a WebSphere Application Server. Make sure to replace the xx with a number to identify your instance.

```
/QIBM/ProdData/WebASAdv/bin/crtnewinst -instance WASxx -bootstrap 9xx -lsd 90xx
```

The 9xx is the WebSphere Administrative port and the 90xx is the Location Service Daemon (LSD).

Wait for the script to complete. A dollar sign (\$) appears when the script completes.

3. This command runs a script that creates a directory structure within the OS/400 Integrated File System (IFS). You will find the directories for your new WebSphere Application Server instance under the directory /QIBM/UserData/WebASAdv/WASxx.

A SQL collection called *WASxxREP* is also created. The directory created is used to store all the objects necessary for your new WebSphere Application Server instance and the SQL collection contains all the configuration parameters.

4. To start your new WebSphere Application Server instance, enter the following command in the QShell environment:

```
/QIBM/ProdData/WebASAdv/bin/strwasinst -instance WASxx
```

Allow several minutes for your new WebSphere Application Server instance to start.

5. Verify the WebSphere Application Server instance started correctly by typing the following OS/400 command:

```
WRKACTJOB SBS(QEJBSBS)
```

You should see the jobs WASxxMNTR and WASxxADMN for your WebSphere instance.

## 2.4 Using the code examples

Part 2, “Programming topics” on page 149, of this redbook introduces a number of code examples. Enough detail has been included in each section to allow you to build the examples shown. A complete listing of all code can be found in Appendix A, “Code examples” on page 265. To download and install the sample code, refer to Appendix B, “Additional material” on page 295.



# Part 1

# Administration topics

Part 1 of this redbook covers the administrative aspects of integrating Domino and WebSphere on the iSeries server. Specifically we cover using different HTTP stacks, options for LDAP directories, and how to configure single sign-on between Domino and WebSphere on the iSeries server. This part contains the following chapters:

- ▶ Chapter 3, “HTTP options” on page 17
- ▶ Chapter 4, “Directory sharing using LDAP” on page 57
- ▶ Chapter 5, “Single sign-on” on page 123





## HTTP options

In this chapter, we discuss the options available for HTTP serving in a Domino and WebSphere integrated environment on the iSeries server. The following topics are discussed:

- ▶ Domino HTTP server support
- ▶ iSeries HTTP server support
- ▶ Running Domino and iSeries HTTP servers together

Configuration examples are also included.

## 3.1 Architecture overview

The IBM WebSphere Application Server is IBM's Java-based Web application server. In a multi-tier e-business environment, the WebSphere Application Server is used as the middle tier. It sits between the HTTP server and the business data. The top tier is the HTTP server. This tier handles requests from the Web browser client. The bottom tier is the business data. This tier includes data stored in RDMS such as DB2 and legacy applications. The middle tier is IBM WebSphere Application Server. This tier provides a framework for a consistent architecture link between the HTTP requests and the business data. This is shown in Figure 3-1.

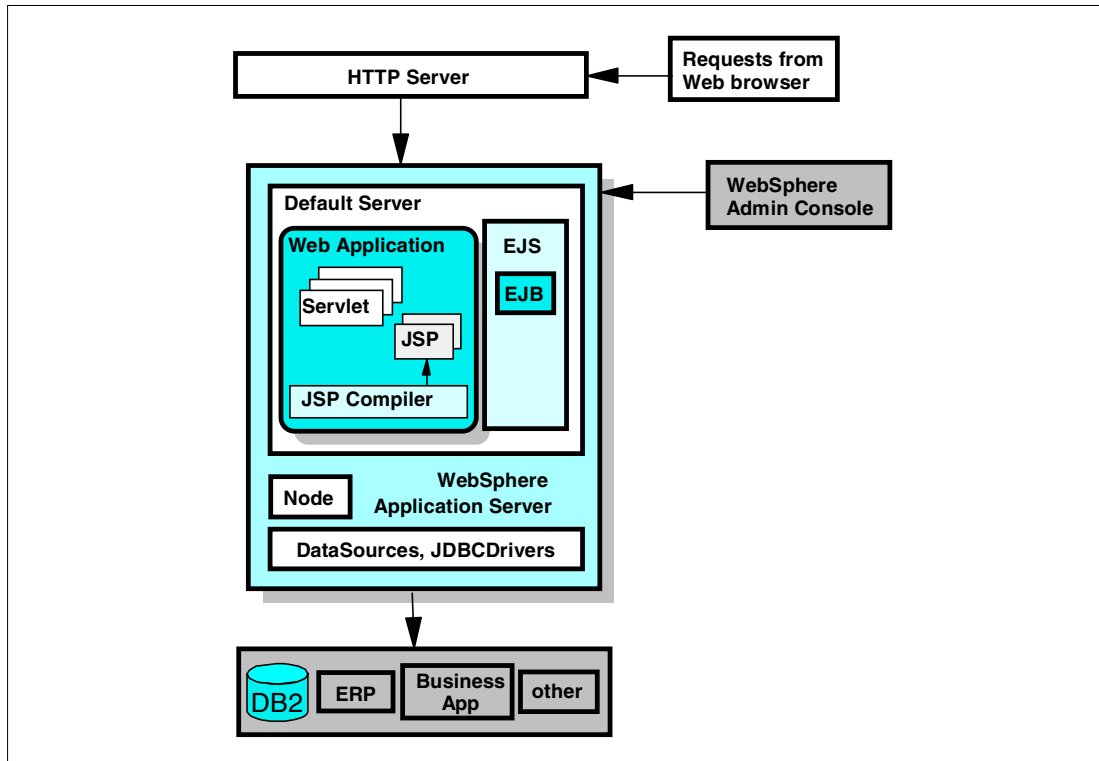


Figure 3-1 Multi-tier e-business environment using WebSphere

On the iSeries server, the WebSphere Application Server for iSeries was based on the IBM HTTP Server for AS/400 (V4R3 or later). The iSeries HTTP server needs to hand the input from a Web form off to the WebSphere Application Server for processing.

There is another powerful tool for sharing information, which is Lotus Domino. The Domino server family is an integrated messaging and Web application software platform. It is suited to growing companies that need to improve customer responsiveness and streamline business processes. One of the services Domino provides is an integrated Domino Web application or HTTP server. As a Web application server, Domino can host Web sites that both Internet and intranet clients can access.

When integrating WebSphere Application Server and Lotus Domino for iSeries, there are two options for Web serving:

- ▶ iSeries HTTP server
- ▶ Domino HTTP server

WebSphere Application Server can use the HTTP server that comes with Domino. Contrarily, Domino can be set up to use the iSeries HTTP server instead of its own internal HTTP server. Regardless of the HTTP server used, iSeries customers can continue to use both Domino and the WebSphere Application Server for different parts of their e-business implementation. The combination of Domino and WebSphere offers customers an opportunity to realize applications that integrate commerce, business processes, and content management capabilities together.

In this chapter we discuss how to configure Domino and WebSphere to use the different HTTP options as outlined in the following sections:

- ▶ Section 3.2, “Domino HTTP server support” on page 19
- ▶ Section 3.3, “iSeries HTTP server support” on page 34
- ▶ Section 3.4, “Running Domino and iSeries HTTP servers together” on page 52

## 3.2 Domino HTTP server support

The IBM WebSphere Application Server is part of an architecture called the Network Computing Framework (NCF). Part of the NCF is a definition of how different Web-enabling components communicate with each other. Therefore, the server API that connects the IBM WebSphere Application Server with an HTTP server is called the NCF Plug-in. It isolates the unique characteristics of each HTTP server in one plug-in rather than imbedding them throughout the IBM WebSphere Application Server. Thus, IBM and others can develop new plug-ins for additional HTTP servers that are independent of new releases of the IBM WebSphere Application Server.

Depending on the platform, WebSphere can plug into a variety of popular Web servers, including the Apache server, IBM HTTP Server, Netscape Enterprise server, Netscape FastTrack server, Microsoft Internet Information Server (IIS), and Lotus Domino R5.

The HTTP server of Lotus Domino for iSeries V5.0.5 or later can be used as the Web server for WebSphere Application Server V3.5.1 or later. The Domino HTTP server can be used for serving both Domino and WebSphere Application Server Web content. This environment is illustrated in Figure 3-2.

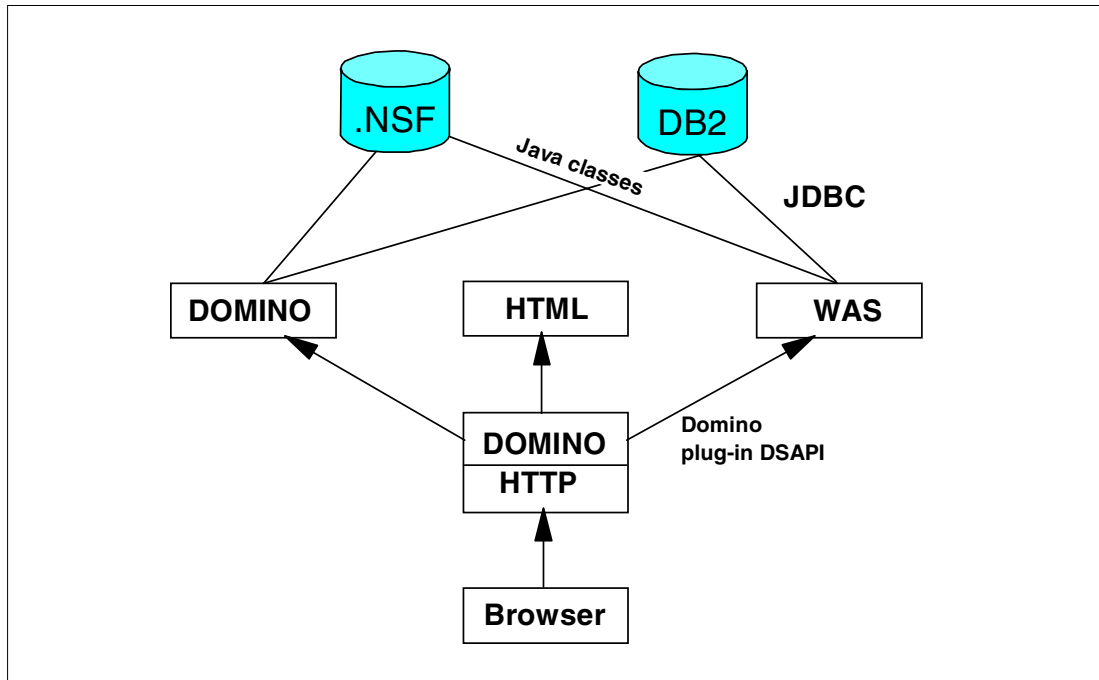


Figure 3-2 Using Domino HTTP server for serving all Web content

For more information on the Lotus Domino for iSeries product, see the Lotus Domino for iSeries Web site at: <http://www.ibm.com/servers/eserver/iseries/domino/>

### 3.2.1 Configuring Domino HTTP server for use by WebSphere

In this section, you learn how to configure the Domino HTTP server for use with the WebSphere Application Server. To do this, perform the following steps:

1. Update the Domino server document.
  - a. From a Lotus Notes client connected to the appropriate Domino server, edit the Domino server document, found in the Domino server's Domino Directory (names.nsf).
  - b. Within in the Domino server document, select the **Internet Protocols** tab and then the **HTTP** sub-tab.
  - c. In the middle of the right column, for the value of the DSAPI filter file names field, enter:  
`/qsys.lib/qejb.lib/domino.srvpgm`

Figure 3-3 shows the DSAPI filter setting in the Domino server document.



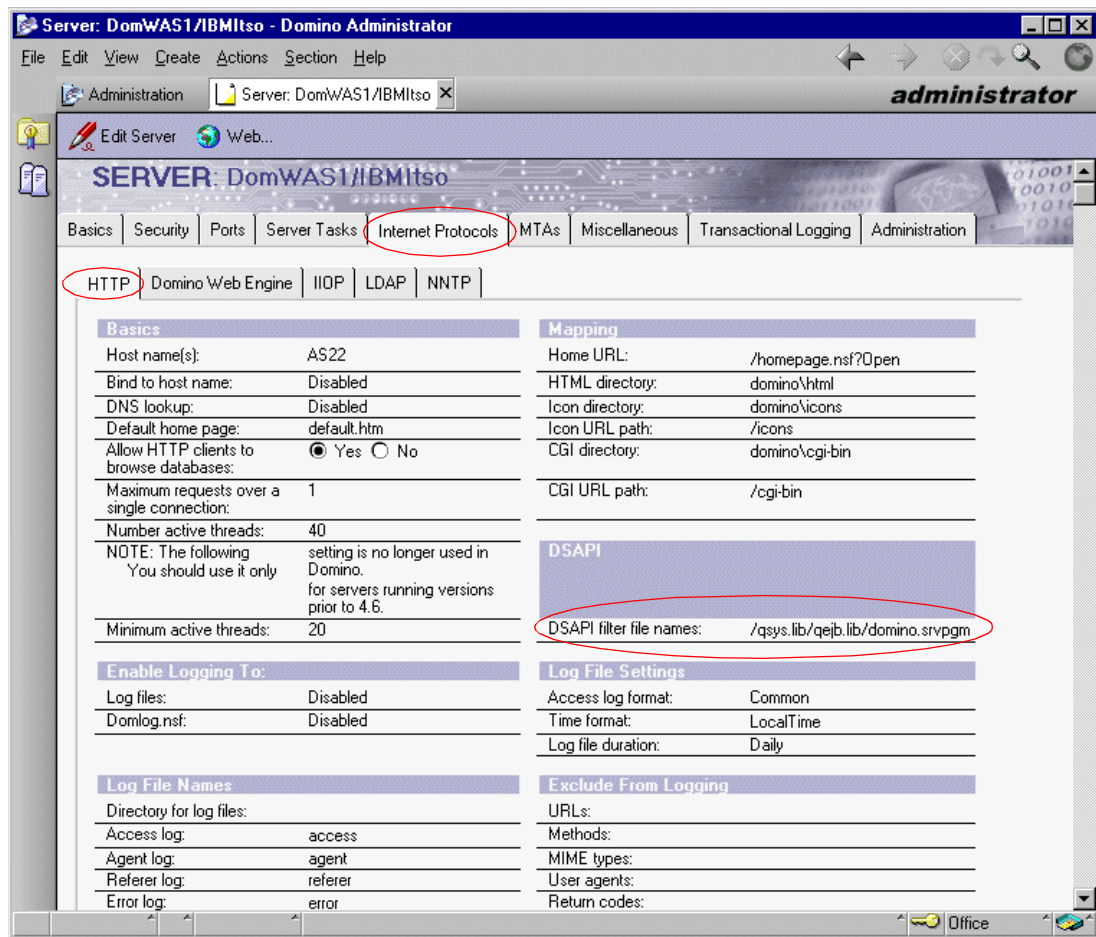


Figure 3-3 Specify DSAPI filter file name in Domino server document

- d. Save and exit the Domino server document. The DSAPI plug-in will be loaded when the Domino HTTP server task is next loaded.

The filter file (domino.srvpgm) specified here is the WebSphere DSAPI plug-in for Domino for iSeries. It passes requests that need WebSphere to the WebSphere Application Server. This behavior is transparent to the Web browser.

**Note:** When you search in the QEJB library on iSeries, you can only find a file named libdomino.srvpgm. But we recommend that you specify domino.srvpgm here, because the Domino server automatically appends the “lib” to the beginning.

We tested using both domino.srvpgm and libdomino.srvpgm for the DSAPI filter file. Neither file name caused errors.

The Domino Server Application Programming Interface (DSAPI) is a C API that lets you write your own extensions to the Domino HTTP server. These extensions or filters let you customize the authentication of Web users. For more information about DSAPI and filters, see the Lotus C API Toolkit for Domino and Notes Release 5.0.3. The toolkit is available at: <http://www.lotus.com/techzone>

2. Update the Domino server's NOTES.INI file. Add the following line to the Domino server's NOTES.INI file:

```
WebSphereInit=/qibm/userdata/webasadv/default/properties/bootstrap.properties
```

For information on how to edit the Domino for iSeries server NOTES.INI, file refer to the redbook *Lotus Domino for AS/400 R5: Implementation*, SG24-5592.

**Note:** The line above is for the WebSphere Application Server's default instance. For a non-default instance, modify the line to reference the directory of the WebSphere Application Server instance. For example, if you have an instance named WAS01, you should change the line as follows:

```
WebSphereInit=/qibm/userdata/webasadv/WAS01/properties/bootstrap.properties
```

The bootstrap.properties file includes some configuration information for the WebSphere DSAPI plug-in. This information allows the plug-in to connect the Domino HTTP server and WebSphere together.

3. Restart the Domino server's HTTP task. Enter the following command from the Domino server console to restart the Domino server's HTTP task:

```
tell http restart
```

4. Update OS/400 authorities. You need to grant the user profile QNOTES the authority needed to be able to create the necessary WebSphere Application Server log files. Enter the following Change Authority (CHGAUT) command from an OS/400 command line:

```
CHGAUT OBJ('/QIBM/UserData/WebASAdv/default/logs') USER(QNOTES) DTAAUT(*RWX)
```

**Note:** The Change Authority command above is for the WebSphere Application Server's default instance. For a non-default instance, modify the OBJ parameter to reference the directory of the WebSphere Application Server instance. For example, if you have a instance named WAS01, you should change the command as follows:

```
CHGAUT OBJ('/QIBM/UserData/WebASAdv/WAS01/logs') USER(QNOTES) DTAAUT(*RWX)
```

### 3.2.2 Using Domino HTTP server to call a WebSphere servlet

The following is an example of using the Domino HTTP server to support requests from a Web browser to access servlets. To follow this example you will need to have configured your Domino server to use the DSAPI filter as described in 3.2.1, "Configuring Domino HTTP server for use by WebSphere" on page 20.

1. Start the Domino HTTP server task. Test that it is functioning correctly by pointing a Web browser to the Domino server. If none of the configuration options used by the HTTP task have been changed, the default Domino server home page is displayed, as shown in Figure 3-4.



Figure 3-4 Default Domino server home page accessed from Web browser

2. Create and start a WebSphere Application Server instance. For details, refer to Section 2.3, "Creating a WebSphere Application Server instance" on page 13.
3. From your desktop start the WebSphere Administrative Console. Expand the Node (in our example shown in Figure 3-5, this is AS22) and select the Default Server. Using the Start icon, start the Default Server. Figure 3-5 shows the WebSphere Administrative Console with the Default Server and Start icon highlighted.

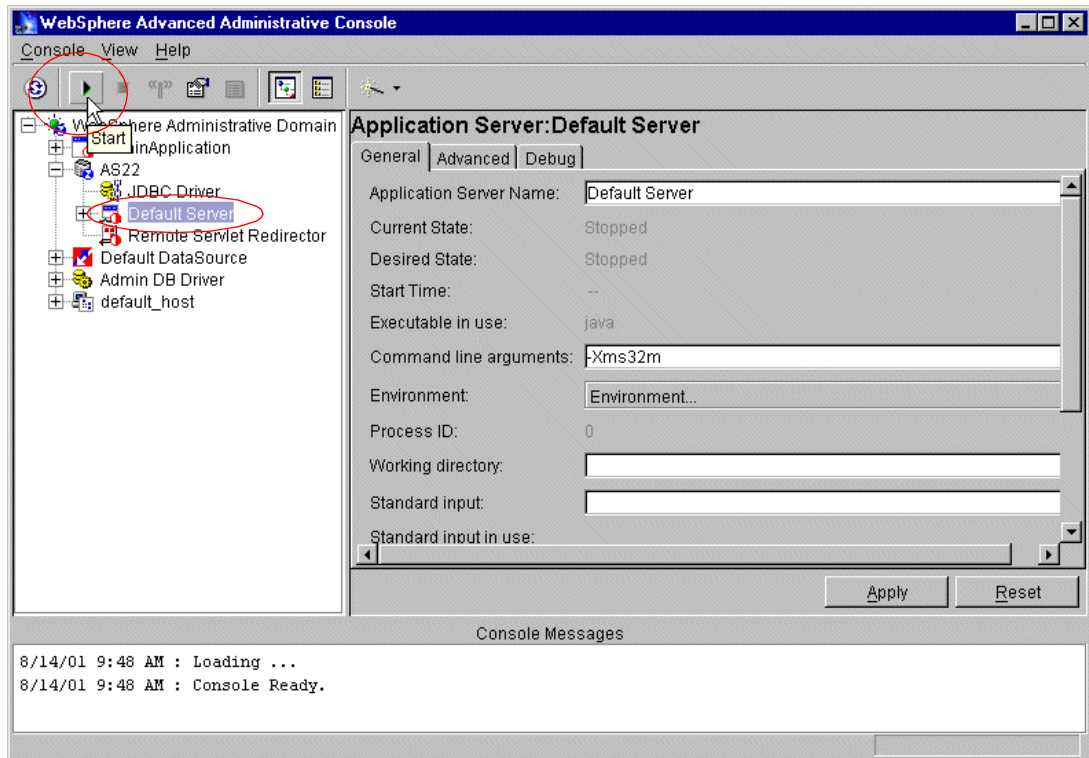


Figure 3-5 WebSphere Administrative Console

The Default Server is created as part of the installation of WebSphere Application Server. It contains a number of sample applications. It is common practice to use these samples to test the installation and configuration of your WebSphere environment. When you create your own applications you should also create a new WebSphere Application Server instance. Although it is possible to deploy your application to the Default Server, it is not recommended.

4. You should now be able access the Snoop servlet. Enter the URL to your Domino server and append `/servlet/snoop`. The result is shown in Figure 3-6.

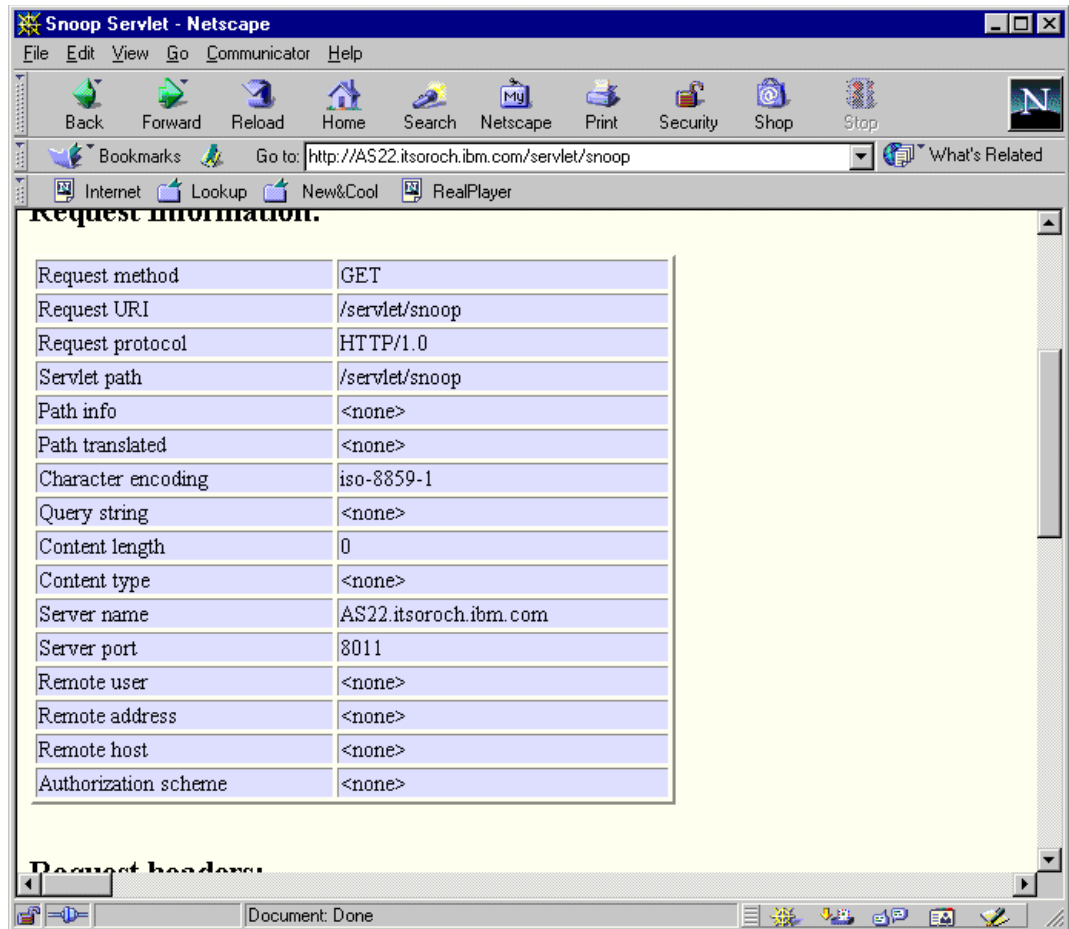


Figure 3-6 Using the Domino HTTP server to call a WebSphere servlet

### 3.2.3 Using Domino HTTP server to manage multiple Web sites

This section discusses the options available for managing multiple Web domains when using the Domino HTTP server and WebSphere Application Server.

When using the Domino HTTP server, each Domino server is associated directly with one WebSphere instance. When using a Domino partitioned server, each Domino server created must be associated with one WebSphere instance.

Creating multiple Domino servers and multiple WebSphere instances can be very expensive in terms of hardware and software. Before configuring your system as such you should consider the following features of both WebSphere and Domino. You may find that your requirements can be met with a less intensive hardware and software solution.

#### WebSphere Application Servers and Web applications

A WebSphere instance can host multiple servers and applications. When you install WebSphere a default server is created for you. The Default Server actually contains multiple Web applications. Figure 3-7 shows the Web applications that are included as part of the Default Server. As can be seen, there is a Web application called “default\_app”, and another called “admin”. Others are also listed. Each application server and Web application can contain its own properties.

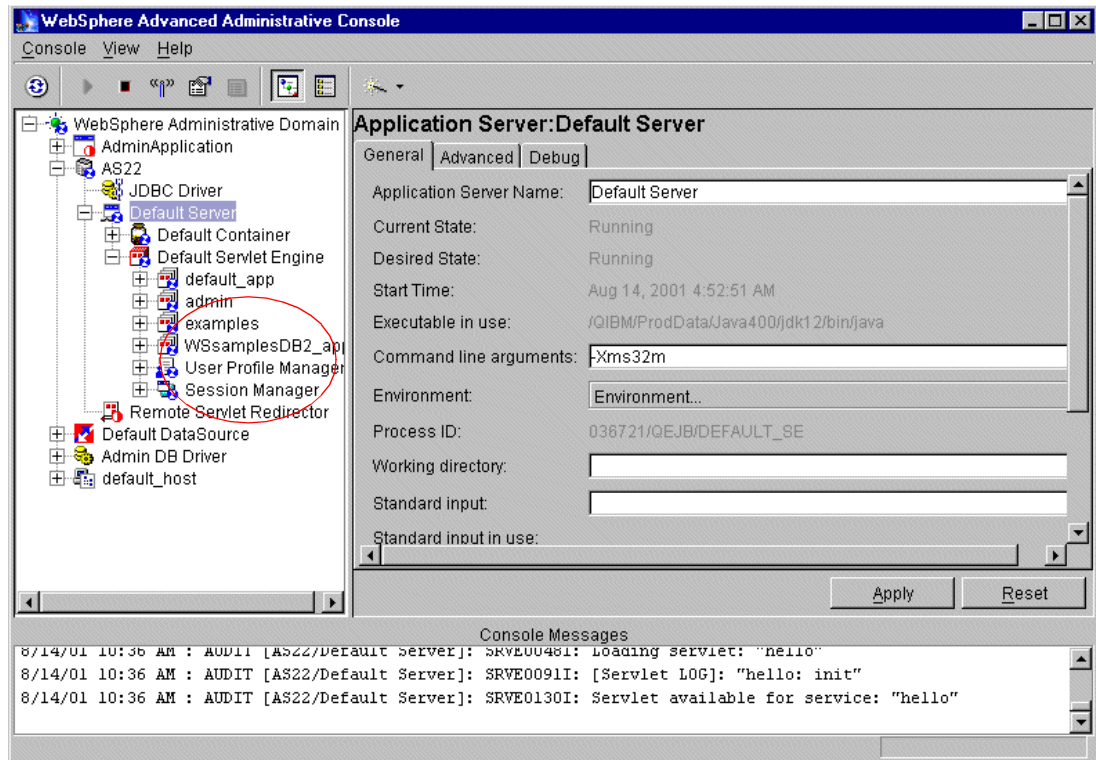


Figure 3-7 Multiple Web applications in the Default Server

If you need to support multiple customers, and the requirement is to host multiple applications, you can create multiple application servers and Web applications. You do not need to create a new WebSphere instance to host each application. For further information on configuring WebSphere, refer to the redbook *WebSphere V3.5 Handbook*, SG24-6161.

## Domino virtual hosts

Domino supports multiple Web sites by using virtual servers. A virtual server is a Web site that is hosted by a Web server managing multiple Web domains. Using virtual servers allows you to maintain separate sites without incurring the expense of additional hardware and software.

You can configure each site in Domino with its own IP address, default home page, customized Web server messages, and HTML, CGI, and icons directories. The Domino data directory, however, is not individually configured for each virtual server; it is shared by all virtual servers. To configure the virtual server, use a Virtual Server document in the Domino Directory.

Before configuring the virtual server settings in Domino, you must set up the network connections for each virtual server. A virtual server can be defined in Domino by either having its own separate, permanent IP address, or it can be identified by its domain name. When identified by the domain name, the server is mapped to the same IP address as the Domino server. For a more detailed description of the Domino Web services, refer to the Domino Administration Help (help\help5\_admin.nsf) database.

## Virtual hosts and WebSphere

We have discussed how WebSphere supports multiple Web applications and Domino supports multiple Web sites. By combining these two features together, it is possible to serve multiple Web sites, with each serving different and independent applications. Figure 3-8 shows a possible architecture for such a solution.

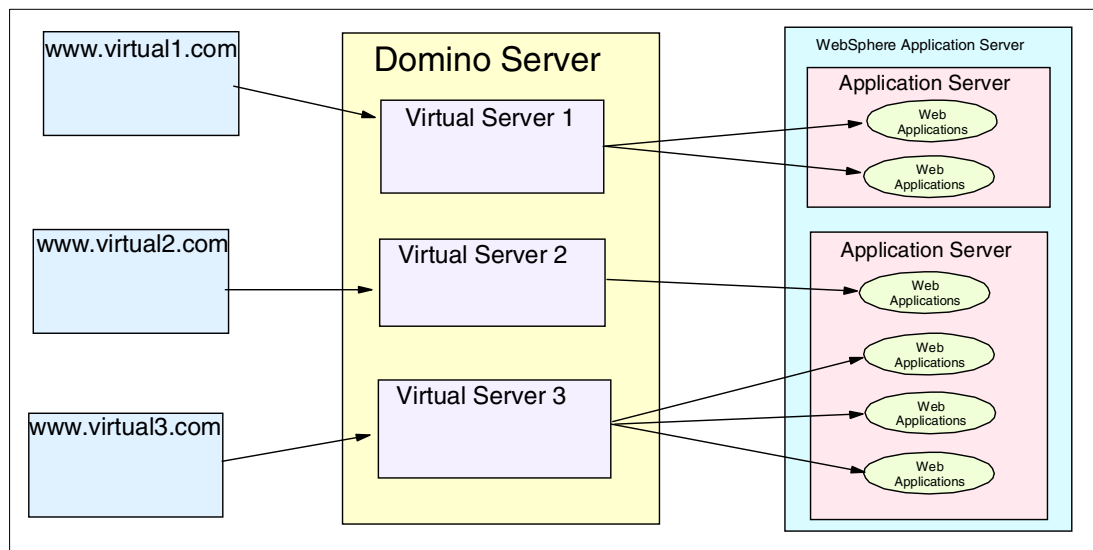


Figure 3-8 Domino virtual hosts and multiple WebSphere Application Servers

## Example of hosting multiple Web sites

In the following example, we configure Domino to support two Web sites on the same physical machine. The sites are [www.nicoleitso.com](http://www.nicoleitso.com) and [www.wendyitso.com](http://www.wendyitso.com).

### Setting up multiple Web sites on one Domino server

To establish a new site on a multi-site Domino server, create a Virtual Server document in the Domino Directory. The settings that you specify in the Virtual Server document override the corresponding settings in the Domino server document. Domino displays the Virtual Server document as a response document to the Domino server document.

To set up a virtual server, perform the following steps:

1. From the Domino Administrator client, click the **Configuration** tab and choose **Server**.
2. Select the Domino server document for the Domino server on which you want to create a virtual server.
3. Click **Web** and choose **Create Virtual Server** from the drop-down list as shown in Figure 3-9.



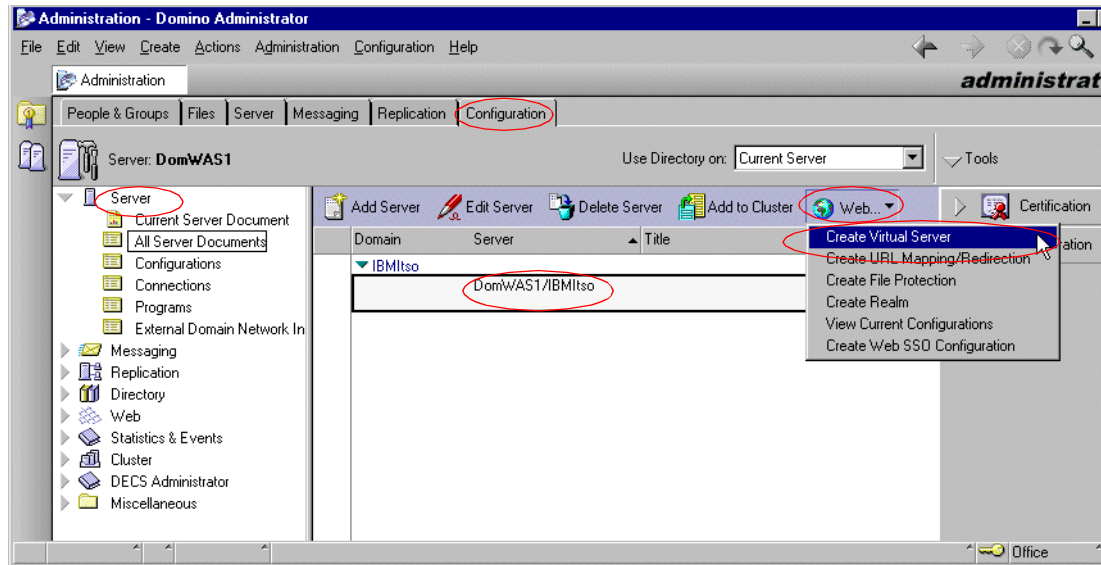


Figure 3-9 Create virtual server

4. On the Virtual Document Type window, choose **Virtual Server** as shown in Figure 3-10. Click **OK**.

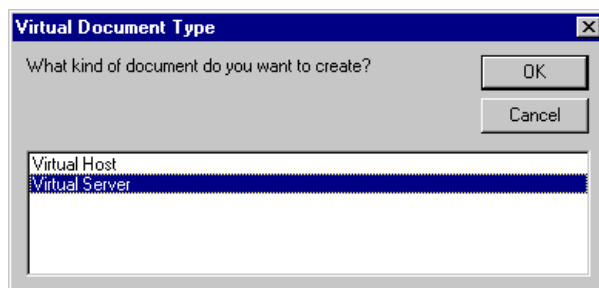


Figure 3-10 Virtual Document Type window

5. On the Virtual Server document, click the **Basics** tab and complete the fields shown in Figure 3-11. The example shown is a Virtual Server document for the DOMWAS1/IBMItso server. We enter `www.nicoleitso.com` in the Hostname field to access the server.



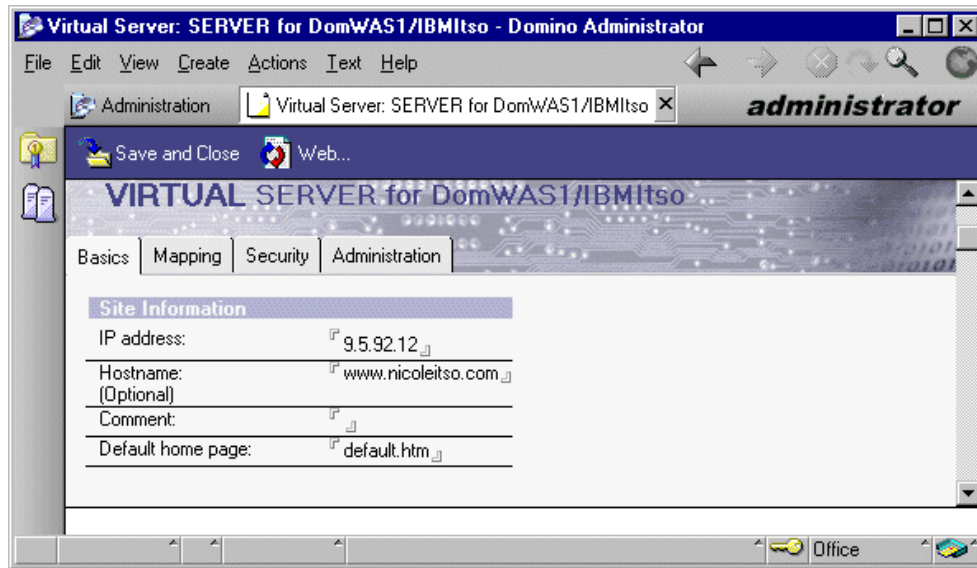


Figure 3-11 Basic tab of the Virtual Server document

6. Click the **Mapping** tab and complete the fields shown in Figure 3-12. Save and close the Virtual Server document.

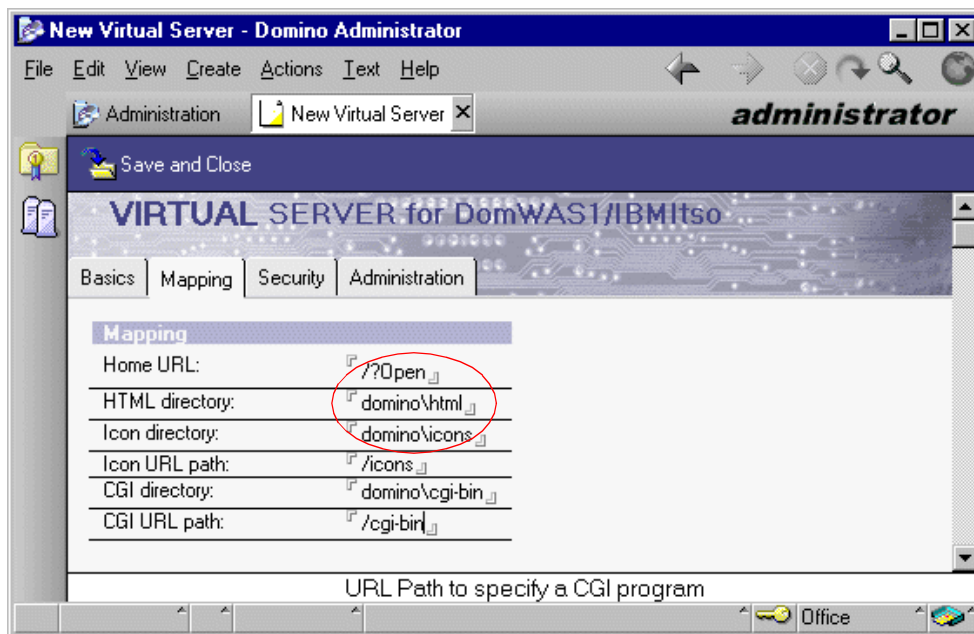


Figure 3-12 Mapping tab of the Virtual Server document

7. Repeat steps 2 to step 6 and create a virtual server for `www.wendyitso.com`. In step 6, change the Home URL field to `/DomWASLab.nsf/?openDatabase`, or any other URL you want to specify as shown in Figure 3-13.

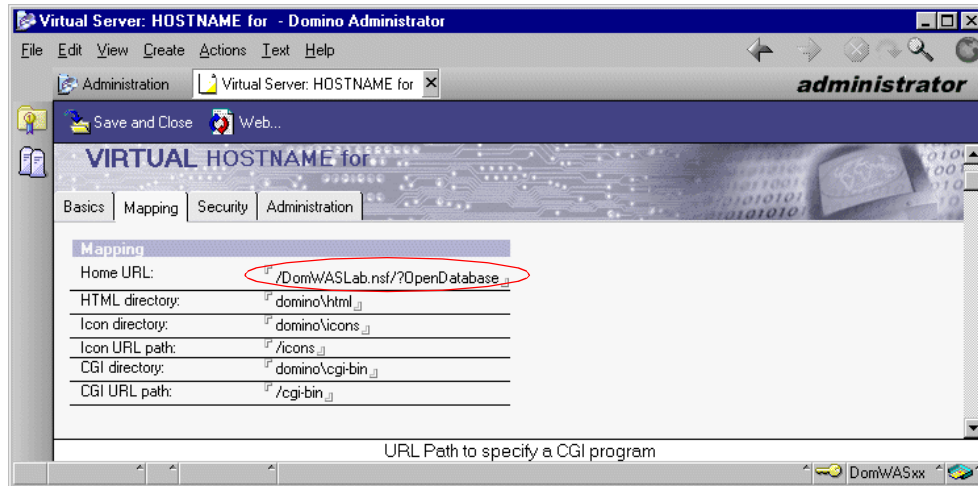


Figure 3-13 Specify different URL for second Virtual Server

8. You need to restart the Domino server's HTTP task. Enter the following command from the Domino server's console so that the settings take effect:

```
tell http restart
```

9. You are now ready to test your new virtual servers. From a Web browser, enter the following URL:

```
http://www.nicoleitso.com
```

The Domino server data directory is displayed as shown in Figure 3-14.

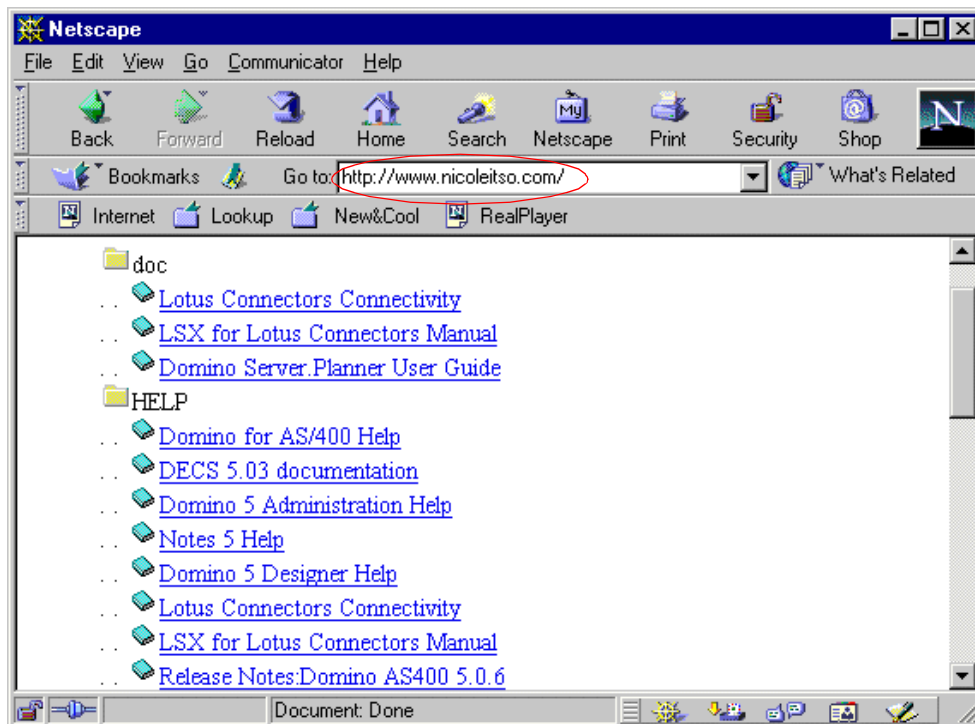


Figure 3-14 www.nicoleitso.com

10. From the Web browser, enter the following URL:

```
http://www.wendyitso.com
```

The Domino WebSphere lab database is displayed as shown in Figure 3-15.

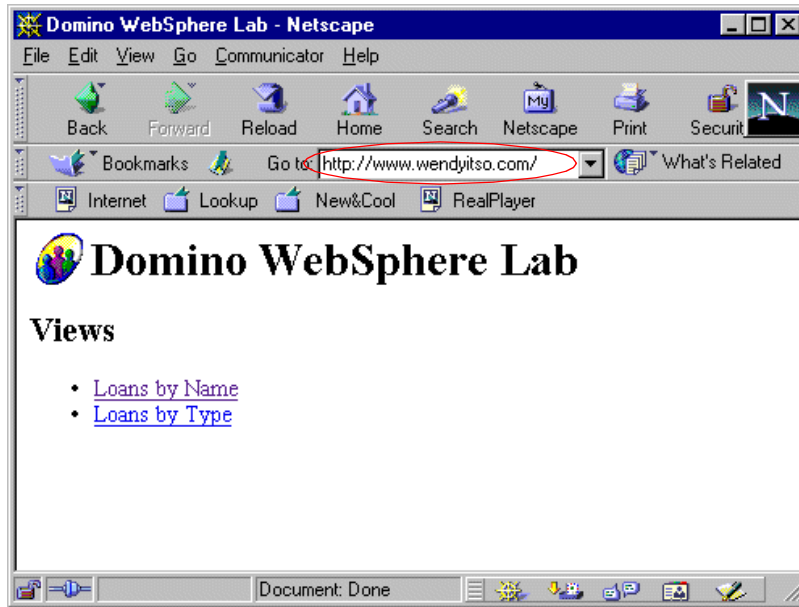


Figure 3-15 www.wendyitso.com

### Using an alias for a Web site

An alias maps multiple DNS names to a single IP address. This allows users to enter different URLs to access a Web site, for example if you want users to be able to enter either `www.nicoleitso.com` or `nicoleitso.com` to access a Web site.

Entering an alias is also useful if you want to change the DNS name for the Web site, for example if the company name changes but you do not want to break links to the previous DNS name. An alias is also known as a virtual host.

To configure a virtual host in Domino, perform the following steps:

1. From the Domino Administrator client, click the **Configuration** tab, choose **Server**, and then select the Domino server document for the Domino server on which you want to create an alias or virtual host.
2. Click the **Web** action button and then choose **Create Virtual Server** from drop-down list.
3. From the Virtual Document Type window, choose **Virtual Host** as shown in Figure 3-16. Click **OK**.

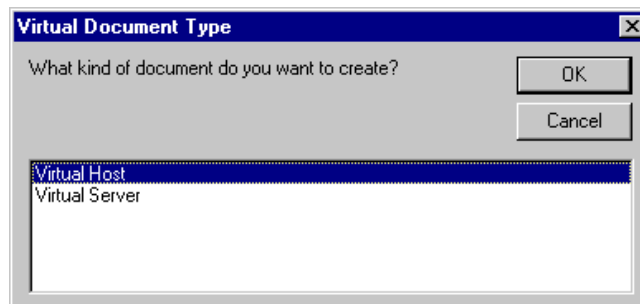


Figure 3-16 Virtual Document Type window

4. On the Virtual HostName document, click the **Basics** tab and complete the fields as shown in Figure 3-17.

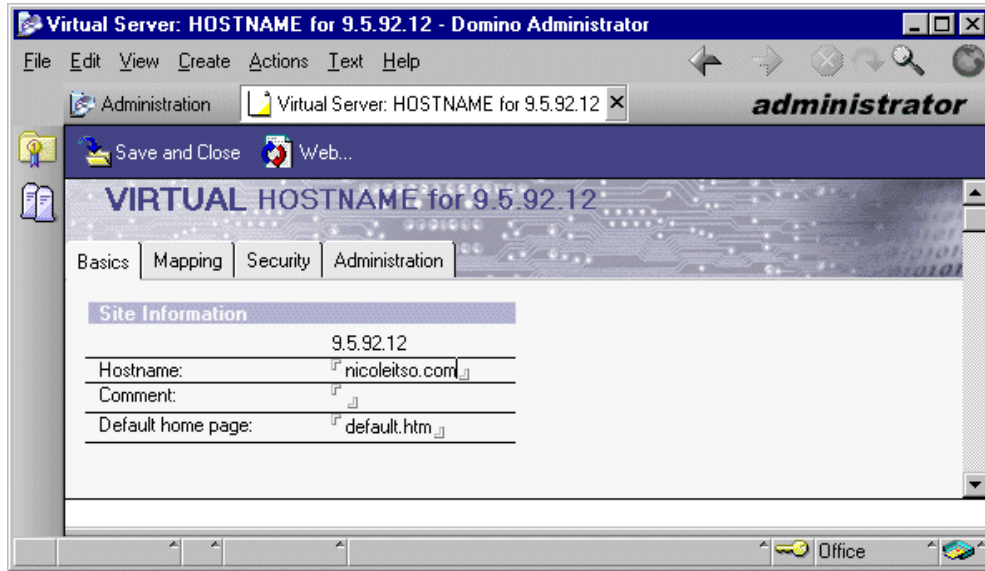


Figure 3-17 Basic tab of the Virtual HostName document

5. Click **Mapping** tab and complete the fields as shown in Figure 3-18. Save and close the Virtual HostName document.

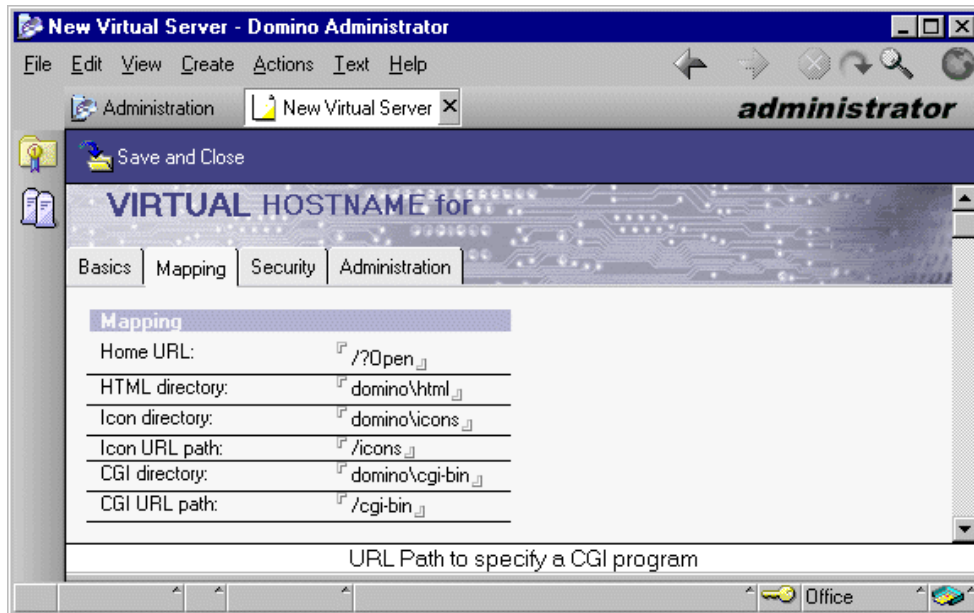


Figure 3-18 Mapping tab of the Virtual HostName document

6. You need to restart the Domino server's HTTP task. Enter the following command from the Domino server's console so that the settings take effect:  

```
tell http restart
```
7. You are now able to test your virtual host. Start your Web browser and enter the following URL:  

```
http://nicoleitso.com
```

You are presented with the same window as seen in Figure 3-14 on page 30.

### ***Creating URL mappings***

URL mappings allow you to control URL and directory mappings. There are three types of URL mappings in Domino:

- ▶ **URL to directory**  
Controls the mapping of a URL to a directory. It allows the administrator to move files or rename directories on the server, but the URL used remains the same. The directory mapping is hidden from the users.
- ▶ **URL to URL**  
Can be used to map URLs from other servers or to create alias mappings for very long URLs. The URL mapping is hidden from users. This allows you to change the server, although the users' bookmarks can remain the same.
- ▶ **URL redirect to URL**  
Very similar to the URL-to-URL mapping. The only difference is the redirected URL is displayed in the Web browser.

To configure a URL mapping in Domino, you must create a URL Mapping/Redirection document in the Domino Directory. To redirect or remap a URL or directory, perform the following steps:

1. From the Domino Administrator client, click the **Configuration** tab.
2. In the Tasks pane, expand the **Server** option and click **All Server Documents**.
3. In the Results pane, select the Domino server that will use the new URL Mapping/Redirection document.
4. Click the **Web** action button and select **Create URL Mapping/Redirection** from drop-down list as shown in Figure 3-19.

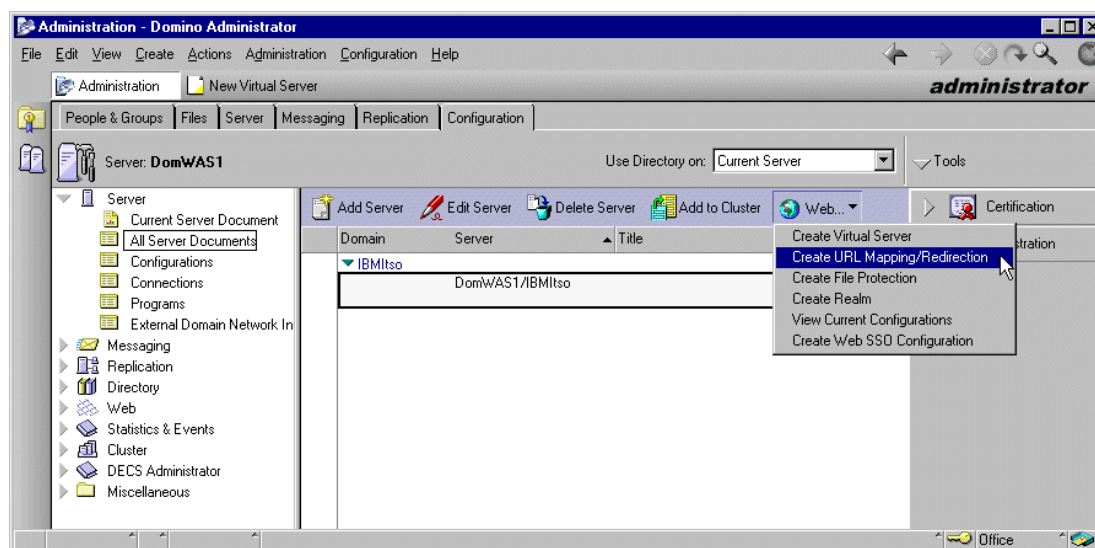


Figure 3-19 Creating a URL Mapping/Redirection document

5. On the Basics tab of the Mapping/Redirection document, select the type of mapping to use, as shown in Figure 3-20.

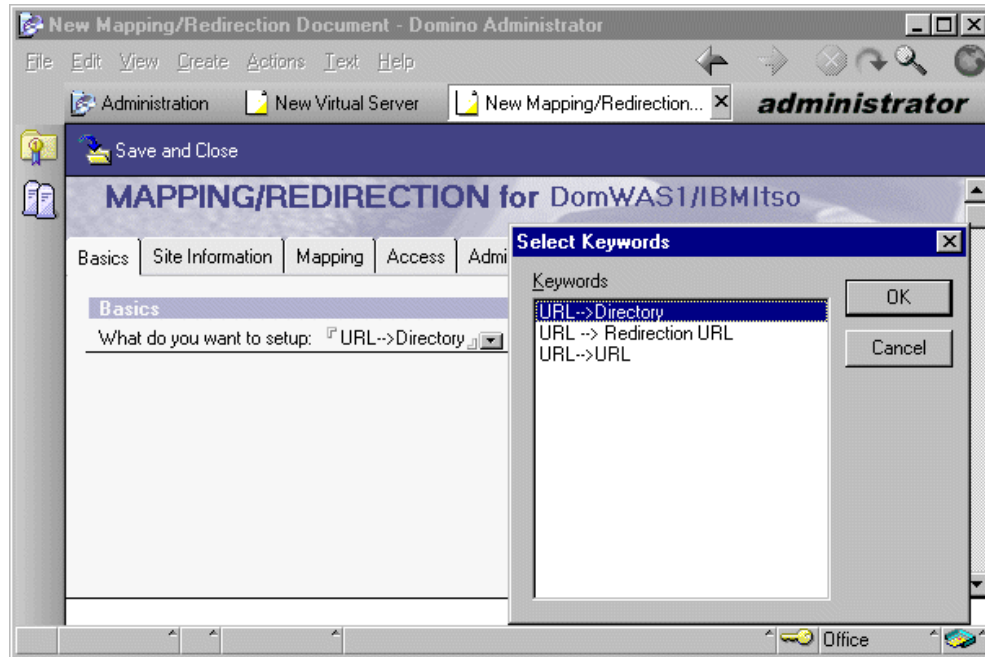


Figure 3-20 Selecting a type of URL mapping to be used

6. Complete the details for the URL Mapping/Redirection. Save and close the document.
7. You now need to restart the Domino server's HTTP task. Enter the following command from the Domino server's console so that the settings take effect:  

```
tell http restart
```

### 3.3 iSeries HTTP server support

The Domino plug-in for iSeries HTTP server allows you to perform iSeries Web serving of Domino and non-Domino content using a single Web server. With this support, Domino can be configured to use the iSeries HTTP server instead of its own internal HTTP server. The architecture is shown in Figure 3-21.

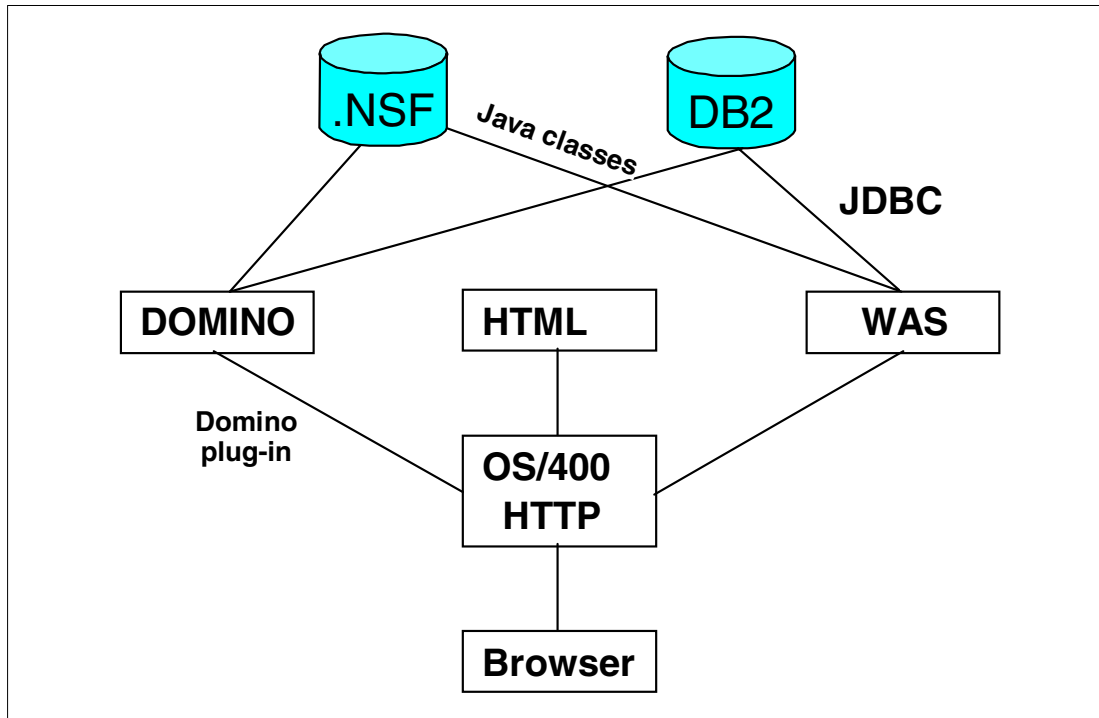


Figure 3-21 Using iSeries HTTP server to serve all Web content

### 3.3.1 Configuring Domino to use iSeries HTTP server

This section describes the steps to enable Domino to use the iSeries HTTP server. You will need to define a one-to-one relationship between an existing Domino server and an existing iSeries HTTP server.

For more information, refer to the redbook *Lotus Domino for iSeries R5: Implementation*, SG24-5592, and the book *HTTP Server for iSeries Webmaster's Guide*, GC41-5434.

During this example, you will need to substitute some values with your specific environment variables. Using Table 3-1 as a guide, you can determine your corresponding values.

Table 3-1 Configuration values to be used in configuring the Domino plug-in for iSeries HTTP server

Values	Description
<iSeries HTTP server name>	The name of the iSeries HTTP server instance to be associated with the Domino server (for example, mywebsvr).
<Domino server name>	The name of the Domino server to be associated with the iSeries HTTP server (for example, mydomsvr).
<Domino server data directory>	The name of the Domino server's data directory (for example, /mydomsvr/domino/data).
<system name>	The internet host name of the iSeries system.

Follow these steps to enable the Domino plug-in for the iSeries HTTP server. Don't forget to replace the tags with the values explained in Table 3-1.

#### **Configuring your iSeries HTTP server instance**

In this section we create and configure an IBM HTTP Server instance to use as our HTTP server for Domino.



1. Start the iSeries HTTP server administration server using the following OS/400 command:  
STRTCPSVR SERVER(\*HTTP) HTTPSVR(\*ADMIN)
2. From a Web browser access the HTTP administration server on port 2001. For example, enter:  
`http://<system name>:2001`  
You are prompted for an iSeries user ID and password. Make sure your iSeries user ID has \*ALLOBJ authority.
3. Click **IBM HTTP Server for AS/400** as shown in Figure 3-22.



Figure 3-22 iSeries HTTP administration server

4. Select the **Configuration and Administration** option from the navigation menu as shown in Figure 3-23.



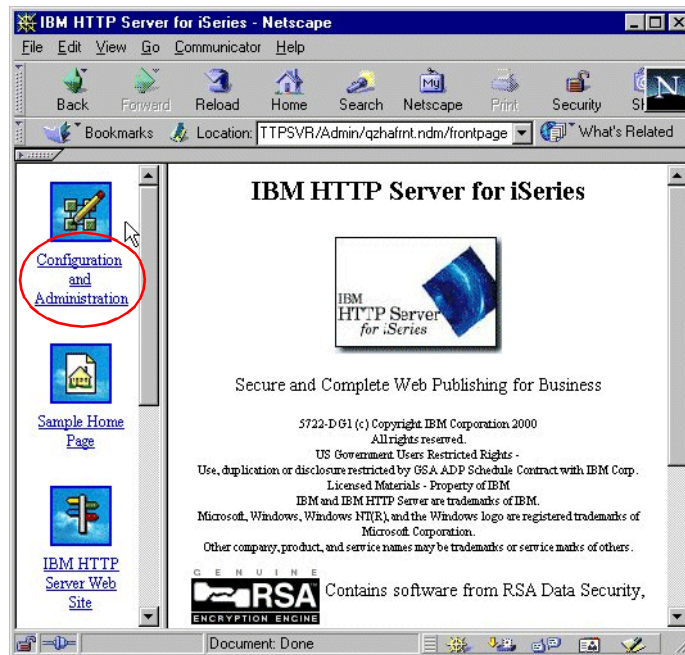


Figure 3-23 IBM HTTP Server for iSeries Web page

- Next we create a configuration for the new HTTP server instance. Click **Configuration** as shown in Figure 3-24.

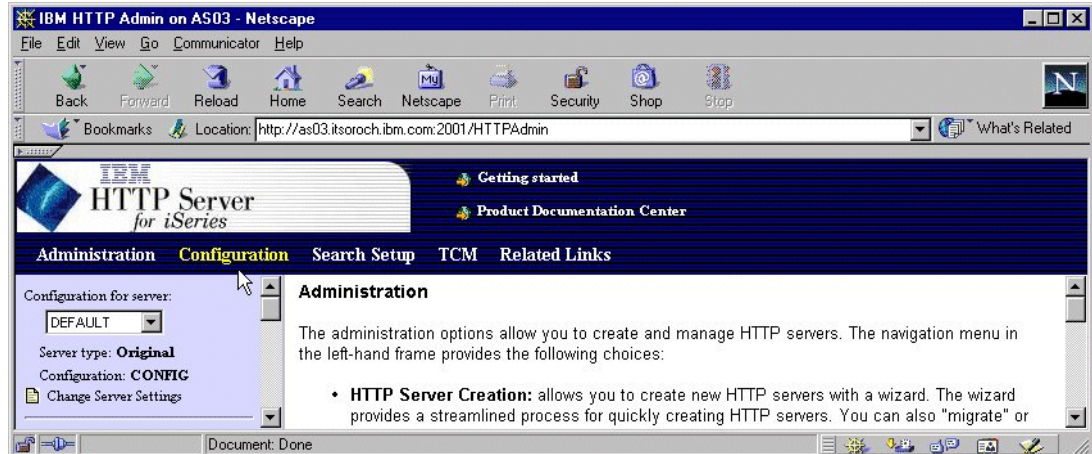


Figure 3-24 HTTP Configuration

- Select the Default server and click **Create configuration**, as shown in Figure 3-25.

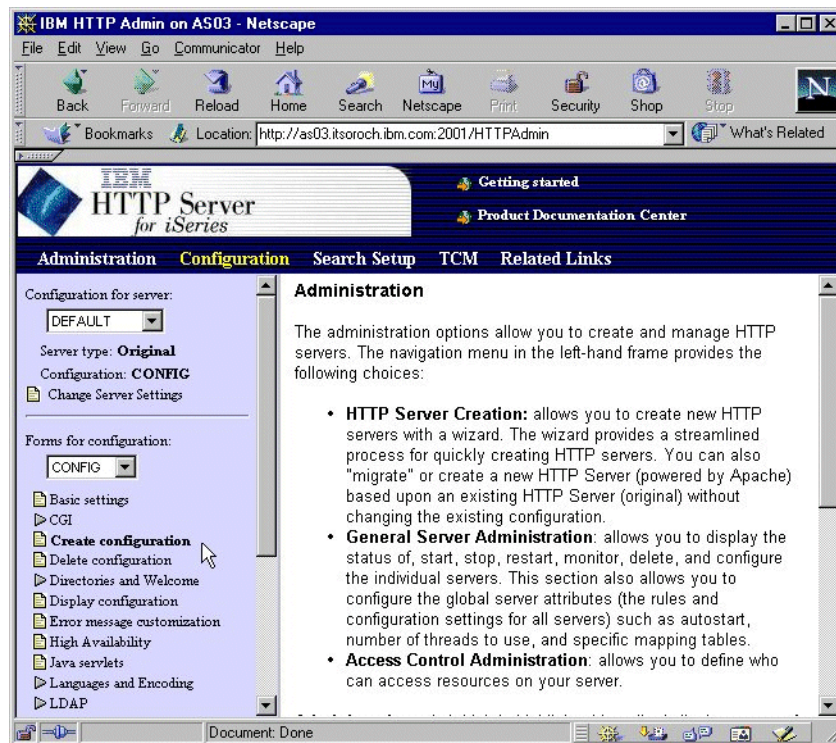


Figure 3-25 HTTP Create configuration

7. The right pane prompts you for a configuration name. This could be anything, but for the purpose of this example we use *DOMWASxx*. Leave the **Create empty configuration** radio button selected and click **Apply** as shown in Figure 3-26.

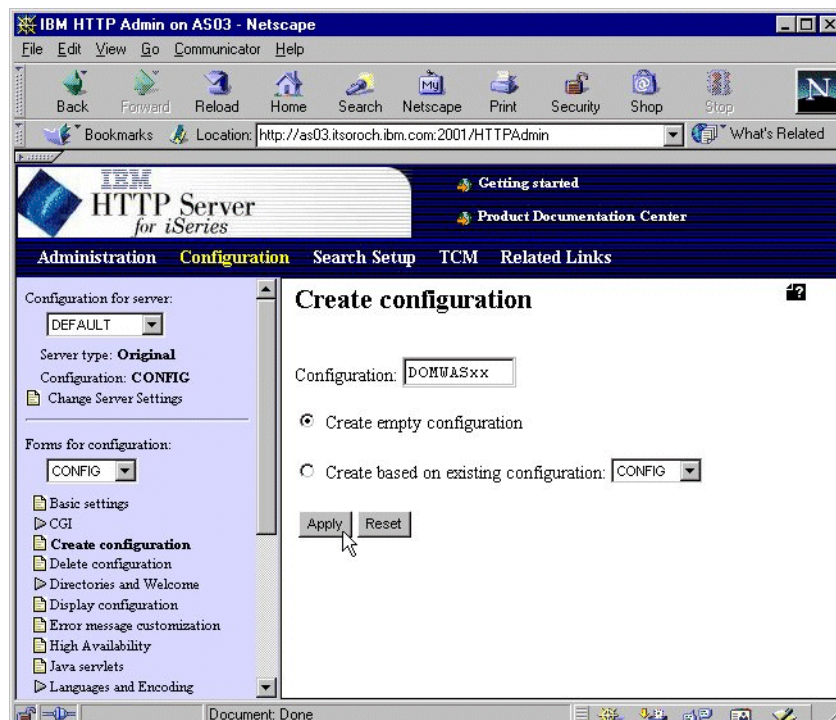


Figure 3-26 HTTP create configuration name

**Note:** Remember to click **Apply** whenever you make changes in this environment or the changes will not be saved.

8. The HTTP server configuration file has been created. Select the name of your configuration in the drop-down box at the left, and click **Basic settings** as shown in Figure 3-27.

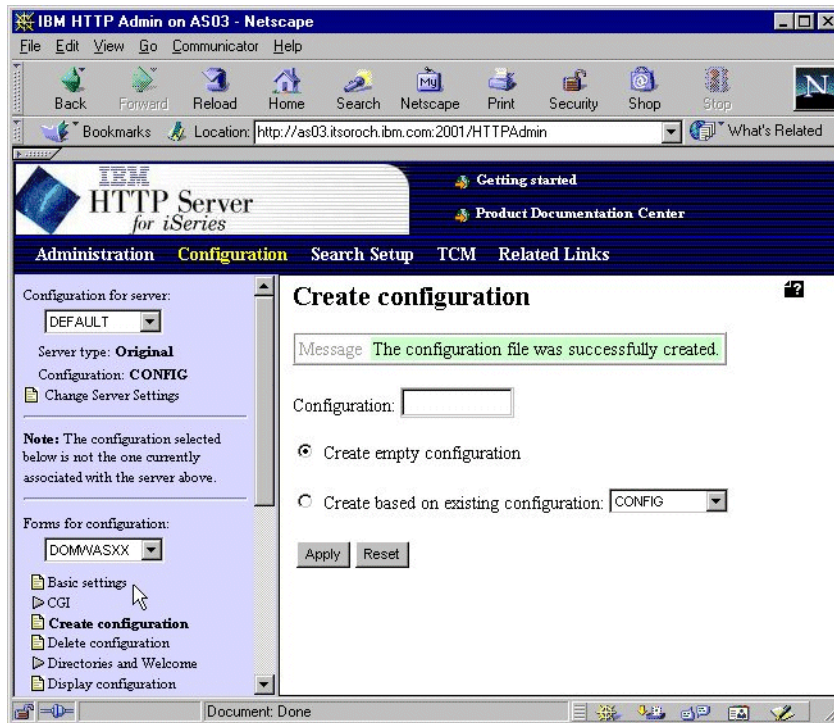


Figure 3-27 HTTP Basic settings

9. In the Host Name box, enter the name of your iSeries server.
10. In the Default Port area, enter the port number for your HTTP server instance. This will usually be port 80.
11. Be sure to click **Apply** so the changes take effect.
12. Enable the HTTP methods required. This allows the HTTP server to process requests and pass them to WebSphere. Click **Request Processing** in the left pane.
13. Click **Methods** and select the boxes for **GET** and **POST** as shown in Figure 3-28. Click **Apply**.



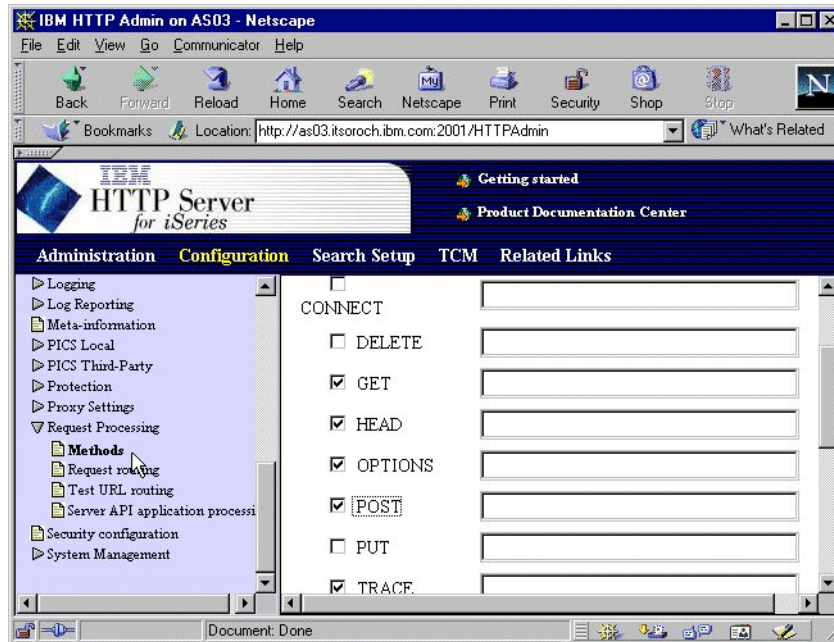


Figure 3-28 HTTP Request Processing: Methods

14. Add support for Java servlets and JSPs. Click the **Java Servlets** link in the left-hand pane as shown in Figure 3-29.

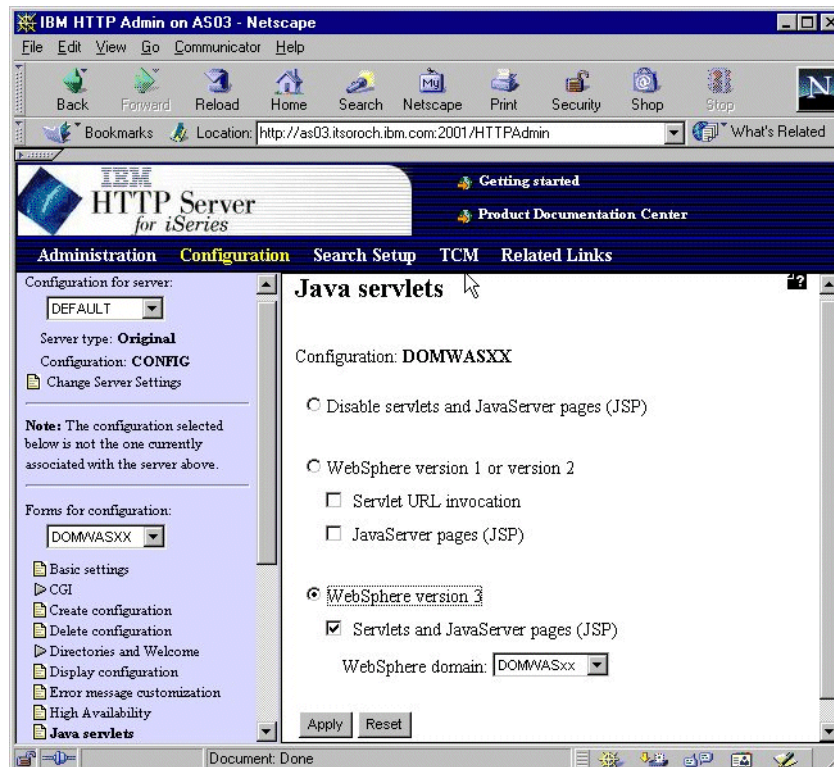


Figure 3-29 HTTP Java servlets and JSPs support

15. Select the **WebSphere Version 3** radio button and select your configuration from the WebSphere domain drop-down box. Click **Apply**.

16. Click **Request Processing**, then **Request Routing**. Add the entries as shown in Table 3-2.

Table 3-2 Request Routing settings

Action	URL template	Replacement file path	CGI conversion mode (in/out)
Pass	/html/*	/QIBM/userData/WebASAdv/<INSTANCE>/html/*	
Service	*.nsf*	/QSYS.LIB/QNOTES.LIB/LIBHTTPX.SRVPGM:Service	%%BINARY/MIXED%%
Pass	/icons/*	<Domino server data directory>/domino/icons/*	
Pass	/domjava/*	<Domino server data directory>/domino/JAVA/*	

**Note:** The settings here assume your Domino server uses the default directories for its icons and Java applets.

Figure 3-30 shows the Request Routing settings in the HTTP server instance.

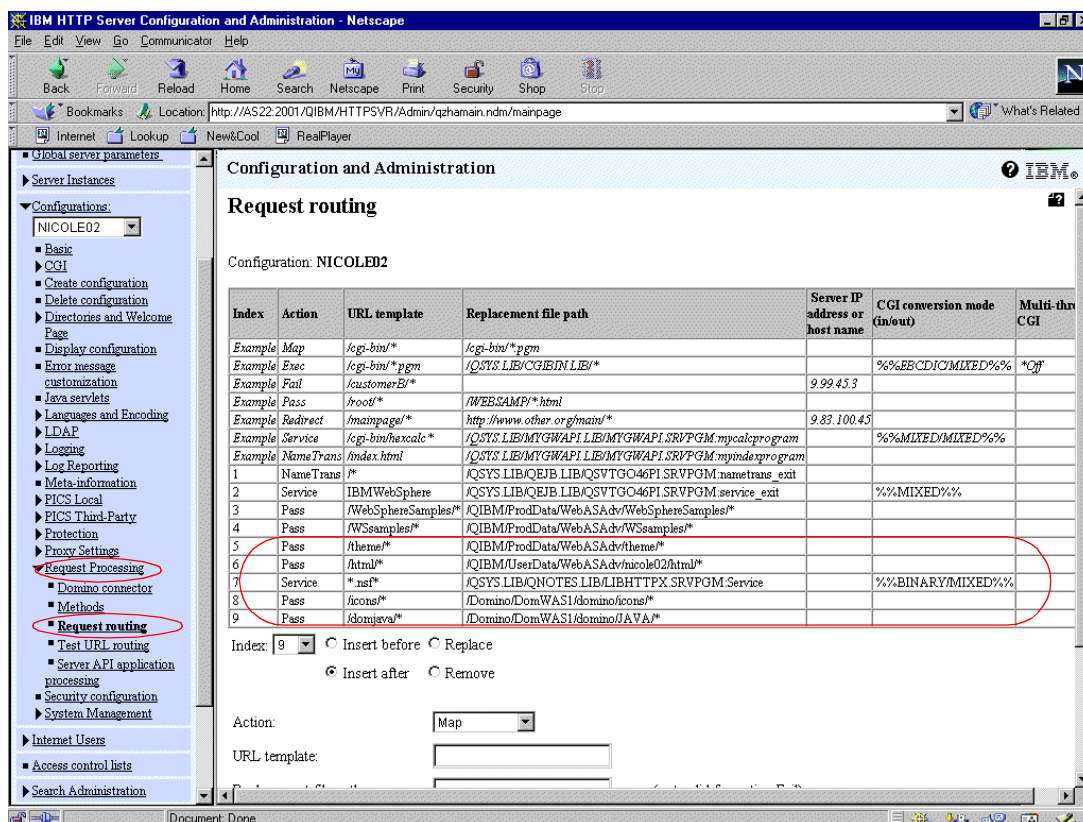


Figure 3-30 HTTP Request Processing: Request Routing settings

17. To complete the configuration, click **Server API Application Processing** from the navigation pane on the left side of the window and add the two lines shown in Table 3-3.

Table 3-3 Server API application processing settings

Step	Application path and file name
ServerInit	/QSYS.LIB/QNOTES.LIB/LIBHTTPX.SRVPGM:ServerInit
ServerTerm	/QSYS.LIB/QNOTES.LIB/LIBHTTPX.SRVPGM:ServerTerm

Figure 3-31 shows the Server API application processing settings in the HTTP server instance.

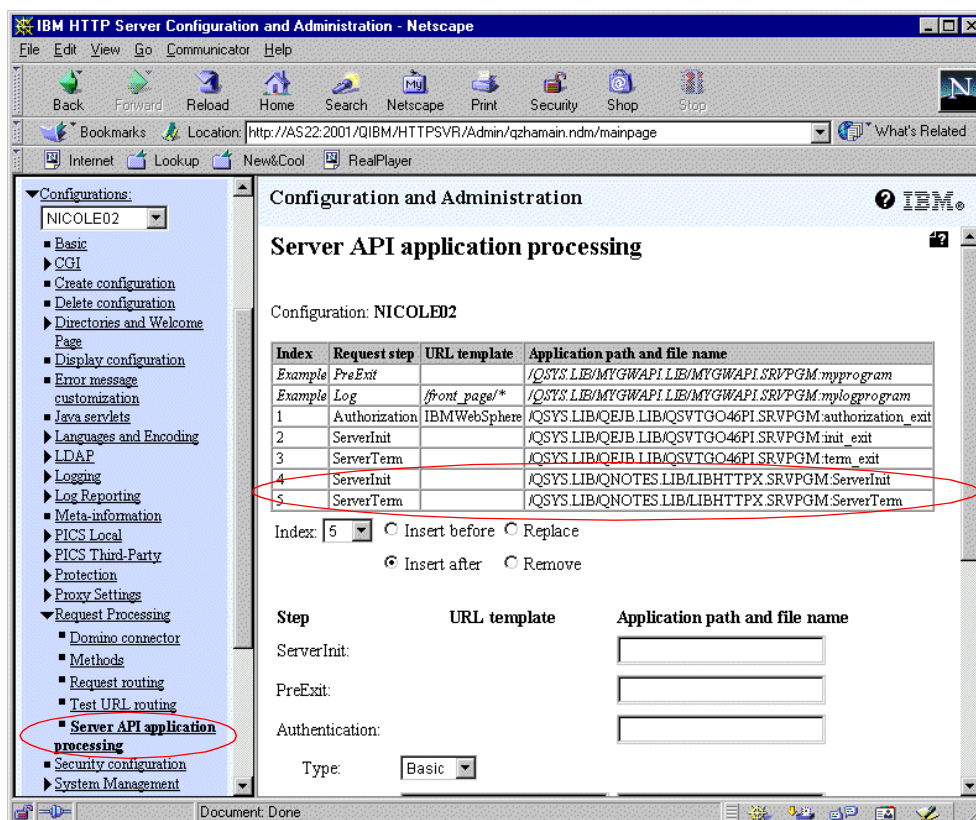


Figure 3-31 HTTP Request Processing: Server API application processing settings

18. Create an HTTP server instance to use this configuration. From your Web browser window, click **Administration**, then **Create HTTP server** and when asked about the type of server, select **HTTP Server (original)** and click **Next** as shown in Figure 3-32.

**Note:** Domino does not support the HTTP server (powered by Apache).



Figure 3-32 Creating an HTTP server instance

19. Give your HTTP server instance a name, for example *DOMWASxx*, and click **Next**.
20. Select **Yes, configure based on an existing original server**. Select your configuration from the drop-down box, and click **Next** then **Finish**.
21. Click the **Manage HTTP Servers** link or click the **Manage** button.
22. Start your HTTP server instance by selecting it and clicking **Start**.

### Reconfiguring your Domino server

In order for your Domino server to use the iSeries HTTP server, you must first reconfigure your Domino server. Perform the following steps:

1. End your Domino server using the following OS/400 CL command:  

```
ENDDOMSVR SERVER(<Domino server name>)
```
2. Update the Domino server so that it uses the iSeries HTTP server instance instead of its own internal HTTP server. To do so, use the following OS/400 CL command:  

```
CHGDOMSVR SERVER(<Domino server name>)WEB(<iSeries HTTP server name>)
```

**Note:** You should not run the Domino HTTP server task and the IBM HTTP Server task in the same environment. Before starting your Domino server again, you should make sure that the HTTP task has been removed from the NOTES.INI. If you have completed other examples from this book, you may also want to make sure the DSAPI filter is no longer referenced in the Domino server document. For details on the DSAPI filter, refer to Section 3.2.1, "Configuring Domino HTTP server for use by WebSphere" on page 20.

3. Start your Domino server using the following OS/400 CL command:  

```
STRDOMSVR SERVER(<Domino server name>)
```



4. If you have not done so already, start your iSeries HTTP server instance using the following OS/400 CL command:

```
STRTCPSVR SERVER(*HTTP) HTTPSVR(<iSeries HTTP server name>)
```

You can monitor the HTTP server jobs using the Work with Active Job (WRKACTJOB) command:

```
WRKACTJOB JOB job(<iSeries HTTP server name>)
```

You should now be able to access Domino via the iSeries HTTP server.

5. From a Web browser, enter the URL to your iSeries HTTP server and append the homepage.nsf as shown in Figure 3-33.



Figure 3-33 Domino served by the iSeries HTTP server

### 3.3.2 Using iSeries HTTP server to manage multiple Web sites

IBM HTTP Server for iSeries allows you to create multiple instances of the product. This means you can host numerous Web sites with different host names or port numbers. You can configure WebSphere Application Server to extend support to multiple server instances, and thus extend server-side Java capability to all of your Web sites.



A virtual host is a configuration enabling a single host machine to resemble multiple hosts. It allows a single physical machine to support several independently configured and administered applications. Each virtual host has a logical name and a list of one or more DNS aliases by which it is known. When a servlet request is made, the server name and port number entered into the Web browser are compared to a list of all known aliases in an effort to locate the correct virtual host and serve the servlet. If no match is found, an error is returned to the Web browser.

The WebSphere Application Server provides a default virtual host with some common aliases, such as the machine's IP address, short host name, and fully qualified host name. The alias comprises the first part of the path for accessing a resource such as a servlet. For example, it is `localhost` in the request `http://localhost/yourServlet`.

A virtual host is not associated with a particular node (machine). It is a configuration, rather than a "live object," explaining why it can be created, but not started or stopped. For many users, virtual host creation will be unnecessary because the `default_host` is provided.

As an example of how virtual hosts are used, suppose an Internet Service Provider (ISP) has two customers whose Internet sites are to be hosted on the same machine. The ISP would like to keep the two sites isolated from one to another, despite their sharing a machine.

The ISP could associate the resources of the first company with `VirtualHost1` and the resources of the second company with `VirtualHost2`. Now suppose both company's sites offer the same servlet. Each site has its own instances of the servlet, which are unaware of the other site's instances.

If the company whose site is organized on `VirtualHost2` is past due in paying its account with the ISP, the ISP can refuse all servlet requests that are routed to `VirtualHost2`. Even though the same servlet is available on `VirtualHost1`, the requests directed at `VirtualHost2` will not be routed there.

The servlets on one virtual host do not share their context with the servlets on the other virtual host. Requests for the servlet on `VirtualHost1` can continue as usual, even though `VirtualHost2` is refusing to fill requests for the same servlet.

The administrator can associate the Web paths of resources, such as servlets, Web pages, and JavaServer Pages (JSP) files, with virtual hosts. It is common to say that the resources are "on" the virtual host, even though the virtual host is a configuration, not a physical machine that can hold files.

The Web path of a resource, such as a servlet, is a path by which users can request the resource. For example, an administrator might specify two Web paths for a servlet class named *Snoopy*. This allows users to specify either `http://www.companyname.com/Snoopy` or `http://www.companyname.com/SnoopyTwo` to request the servlet.

Because the administrator associates the Web path of a resource, and not the resource itself, with a virtual host, the administrator can associate one Web path of a servlet with one virtual host, and another Web path of the servlet with a different virtual host. WebSphere provides the flexibility to set up virtual hosting in the way that best suits your needs.

## The default virtual host

WebSphere provides a default virtual host (named *default\_host*). The default uses port 80. The `default_host` has the following aliases:

- ▶ The IP address of the local machine (*yourIPAddress*)
- ▶ The "localhost" alias, meaning the local machine (*localhost*)
- ▶ The host name (such as *server1*)

- ▶ The fully qualified host name (such as *server1.yourcompany.com*)
- ▶ The loopback address (127.0.0.1)

Once in a while, the fully qualified name cannot be constructed. If several paths containing the fully qualified name do not seem to be working, use the WebSphere Administrative Console to check the virtual host's Aliases property to ensure the fully qualified name is registered as an alias.

Unless the administrator specifically wants to isolate resources from one another on the same node (physical machine), he or she probably does not need any virtual hosts in addition to the default host.

When a user requests a resource, WebSphere tries to map the request to an alias of a defined virtual host. The mapping is case insensitive, but the match must be alphabetically exact. Also, different port numbers are treated as different aliases.

## **Multiple IBM HTTP Server for iSeries instances**

You can use one of the following ways to support multiple IBM HTTP Server for iSeries instances:

- ▶ Multiple Domain Name System (DNS) aliases
- ▶ Multiple virtual hosts with a unique alias

### ***Configuring multiple Domain Name System (DNS) aliases***

An alias has the format of hostname:port. You can use the WebSphere Administrative Console to add your multiple HTTP server instance aliases to the WebSphere configuration. This method allows you to use the same WebSphere Application Server configuration and centralized administration for all your HTTP server instances. However, keep in mind that if you use this method, all WebSphere resources are available to all HTTP server instances.

To configure multiple Domain Name System (DNS) aliases, perform the following steps:

1. Start the WebSphere Administrative Console.
2. In the Topology view, select your virtual host (such as default\_host).
3. In the main pane, click the **Advanced** tab as shown in Figure 3-34.

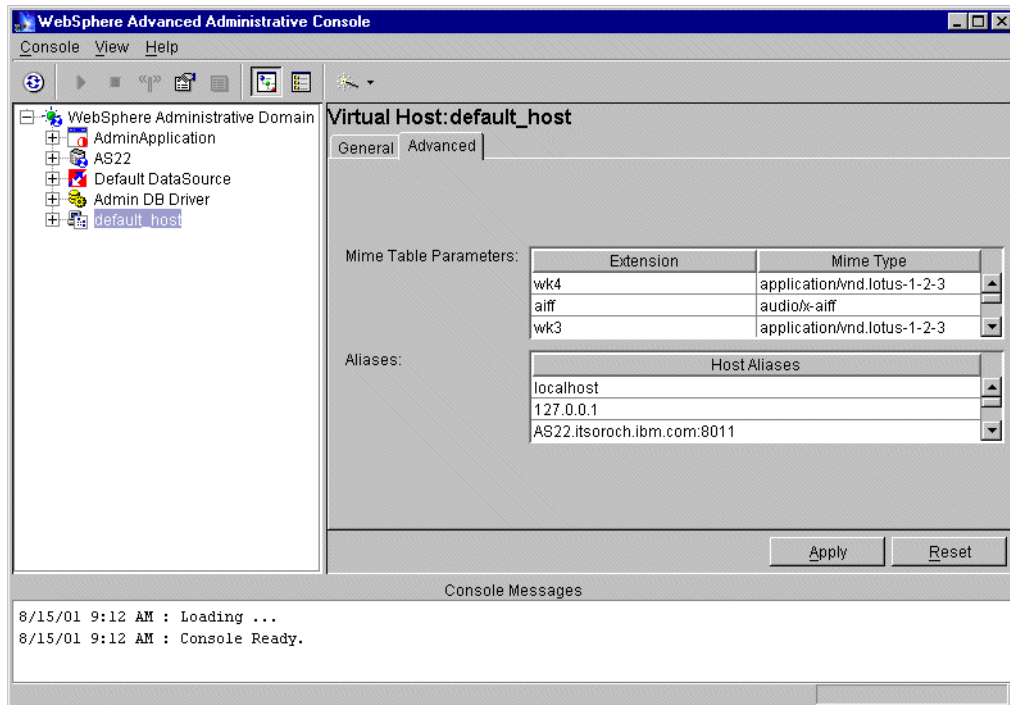


Figure 3-34 WebSphere Administrative Console - Advanced tab for default server

- In the Host Aliases box, scroll down to the first empty line.
- Click the line to get a cursor. Type the desired alias, and press Enter. Click the next line to add the next alias. You should add the following aliases:
  - Localhost and port number: localhost:port
  - Loopback port: 127.0.0.1:port
  - Fully qualified host name and port number: your.host.name:port
  - Short host name and port number: host\_name:port
  - TCP/IP address and port number: x.x.x.x:port
 Repeat this step for each HTTP server instance.
- Click **Apply**, as shown in Figure 3-35.

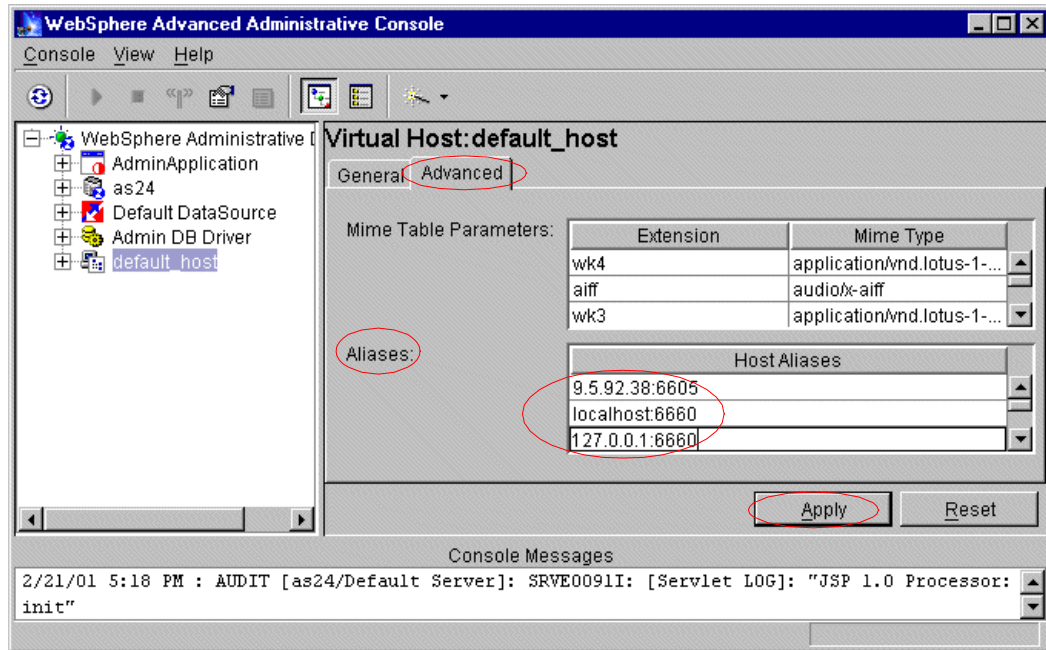


Figure 3-35 WebSphere Administrative Console: Configuring multiple host aliases

7. From the Topology view, right click your default application server to display the pop-up menu, then select **Stop**. After the server is ended, select **Start** from same pop-up menu to restart your WebSphere Application Server, as shown in Figure 3-36.

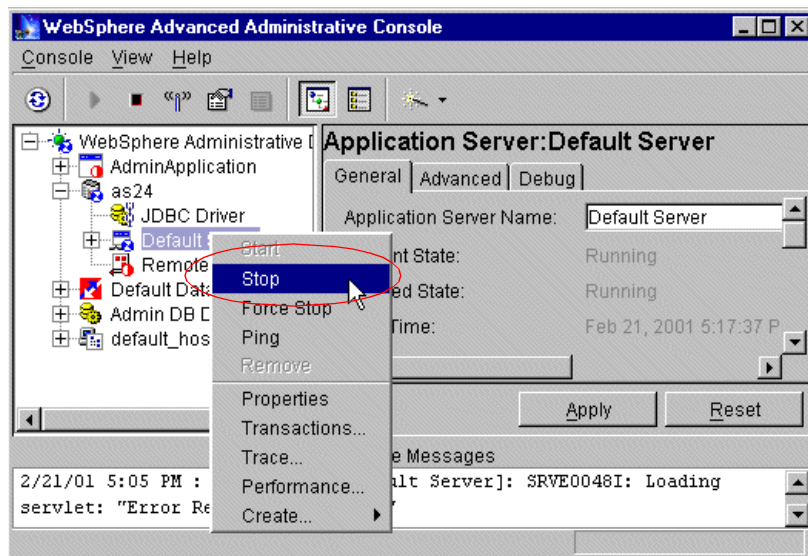


Figure 3-36 Restarting the WebSphere Application Server

### **Configure multiple virtual hosts with a unique alias**

A virtual host is a configuration that enables a single host machine to resemble multiple hosts. Each virtual host has aliases associated with it, so you can configure a virtual host to be associated with a particular host name and port number. This method allows you to separate and control (by URL) which resources are available for client requests.

You can also create multiple instances of the WebSphere Application Server. Each Application Server can have a unique HTTP server instance associated with it. Creating multiple Application Servers allows you to provide isolated development and testing environments. For more information on the WebSphere Application Server running on the iSeries server, visit the following Web site:

<http://www.ibm.com/servers/eserver/iseries/software/websphere/wsappserver/>

To configure multiple virtual hosts with unique aliases, perform the following steps:

1. Start the WebSphere Administrative Console.
2. From the Console menu, select **Tasks**, then select **Configure a Virtual Host** from drop-down menu, as shown in Figure 3-37.

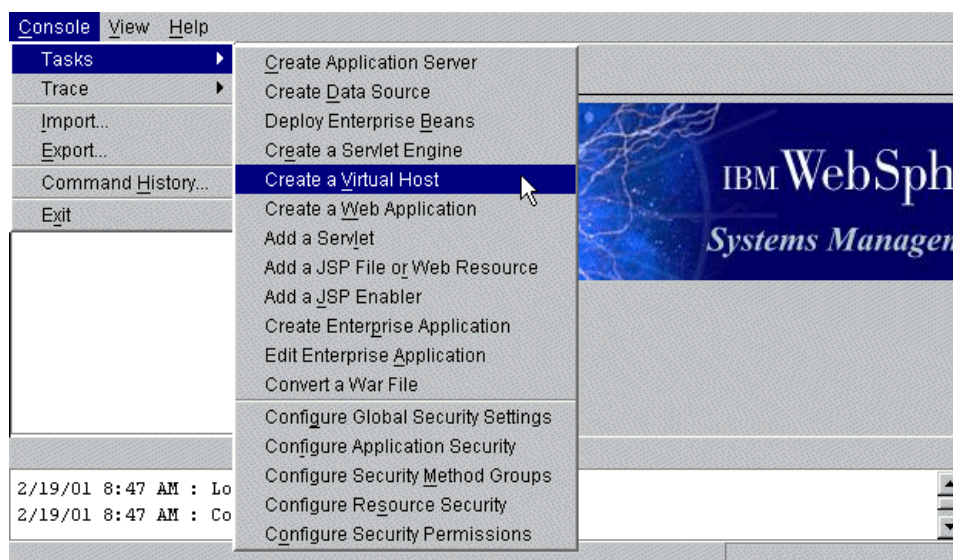


Figure 3-37 Create a WebSphere Virtual Host

3. The Create Virtual Host window appears. Specify a name for your virtual host and click **Next**, as shown in Figure 3-38.

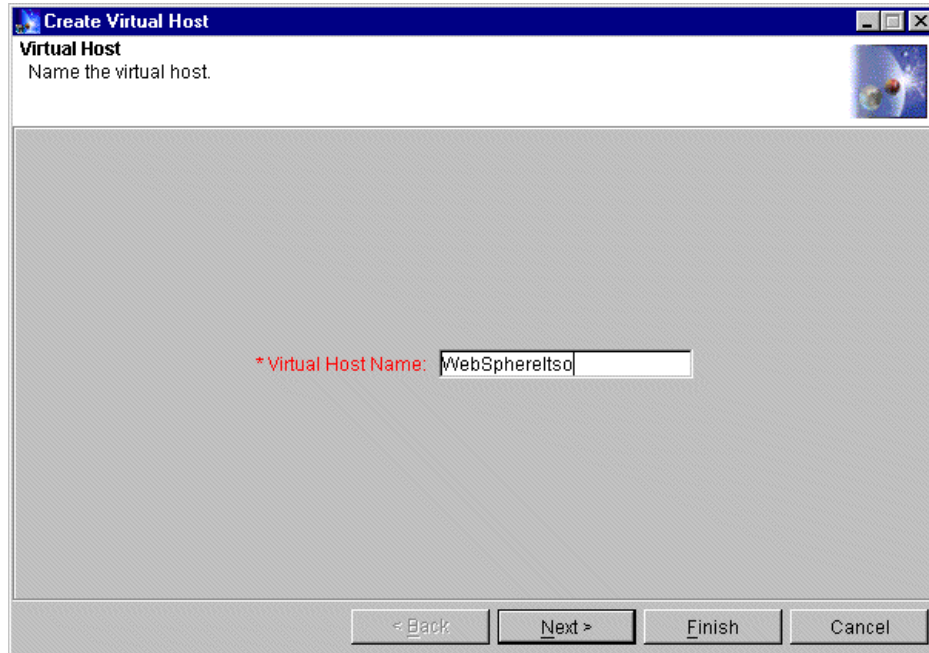


Figure 3-38 WebSphere Virtual Host Name

4. The Virtual Host window appears. In the Host Aliases box, click the empty line to get a cursor. Type the desired alias, and press Enter (Figure 3-39). Click the next line to add the next alias. Add the following aliases:

"localhost" alias and port number:	localhost:port
Loopback address and port number:	127.0.0.1:port
Fully qualified host name and port number:	your.host.name:port
DNS name and port number:	host_name:port
IP address and port number:	x.x.x.x:port

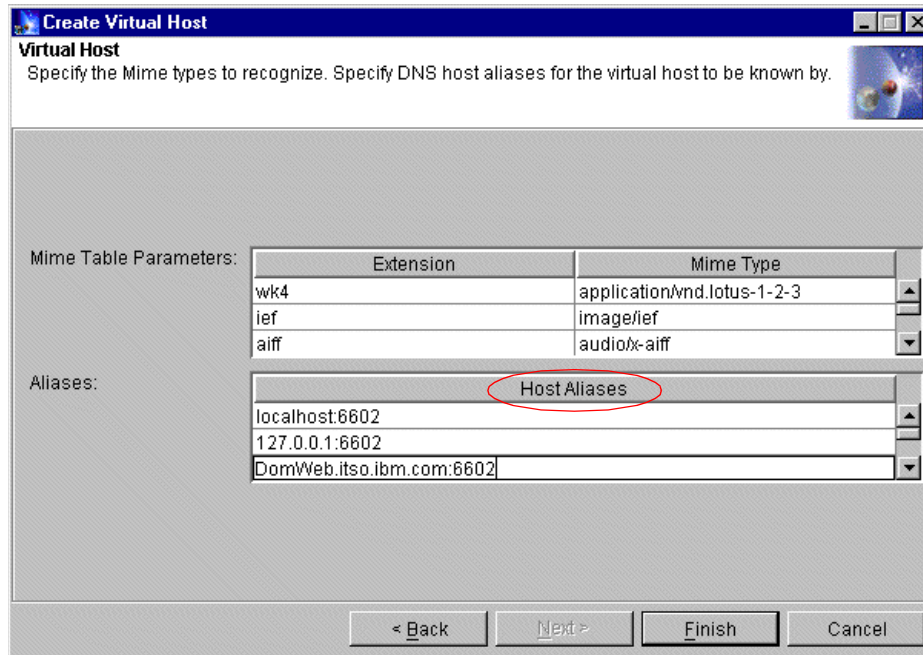


Figure 3-39 Virtual Host Aliases

5. Click **Finish**.
6. Check the topology tree to verify that your virtual host was added in the WebSphere topology.

Now, you can manage your Web resource separately using the alias you set up before with the result similar to Figure 3-40.

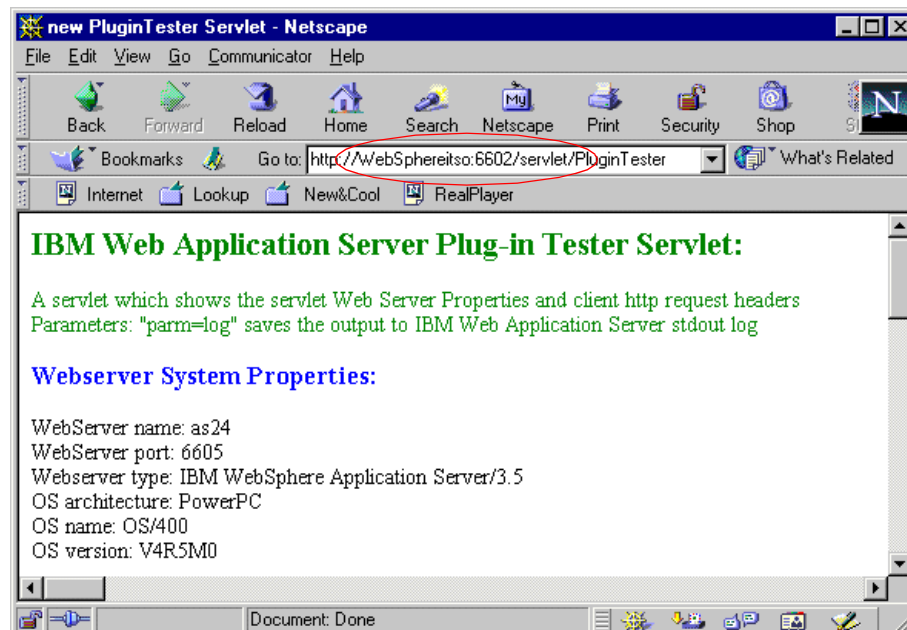


Figure 3-40 Accessing a servlet through Web browser

### 3.3.3 Switching from iSeries HTTP server to Domino HTTP server

If you want to switch from Domino using the iSeries HTTP server back to the Domino HTTP server, perform the following steps:

1. End the Domino server:

```
ENDDOMSVR SERVER(<Domino server name>)
```

2. End the iSeries HTTP server:

```
ENDTCPSVR SERVER(*HTTP) HTTPSVR(<iSeries HTTP server name>)
```

3. Change the Domino server:

```
CHGDOMSVR SERVER(<Domino server name>) WEB(*HTTP)
```

4. Start the Domino server:

```
STRDOMSVR SERVER(<Domino server name>)
```

If you want to still use the iSeries HTTP server to serve non-Domino content, you must comment out the `ServiceInit`, `Service`, and `ServiceTerm` directives in the iSeries HTTP configuration file. See Section 3.3.1, “Configuring Domino to use iSeries HTTP server” on page 35 for details.

## 3.4 Running Domino and iSeries HTTP servers together

It is possible to run both the iSeries HTTP stack and Domino R5's HTTP stack on the same host. However, each HTTP server must use its own TCP port. If you try and start both services using the default port (80), an error will occur on one of the services.

This section provides an example of using the iSeries HTTP stack and the Domino HTTP stack together.

### 3.4.1 Configuring Domino to use a different port for HTTP

Perform the following steps in order to reconfigure the Domino HTTP stack to listen to a different port. In our example we use port 8080.

1. From the Domino Administrator client, open the Domino Directory on the Domino server.
2. Expand the **Server** folder and open the **Servers** view.
3. Click the **Server** document for your Domino server and click the **Edit** button.
4. Select the **Port** tab, then the **Internet Ports** tab, and then the **Web** tab. Change the TCP/IP port number field to 8080. The result is shown in Figure 3-41.



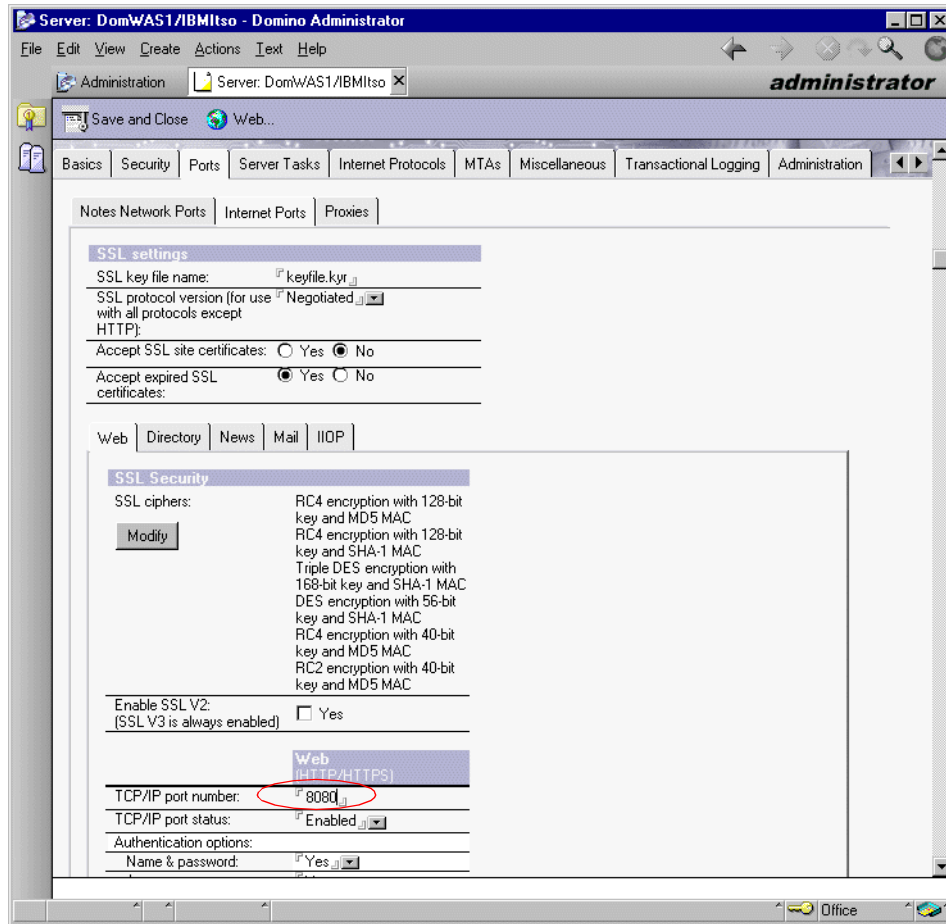


Figure 3-41 Change port number for Domino HTTP server

5. Click the **Save and Close** button to exit the Domino server document.

### 3.4.2 Defining a virtual host for WebSphere to recognize Domino calls

This section describes the steps required to define a virtual host for WebSphere to recognize calls from Domino. Perform the following steps:

1. Start the WebSphere Administrative Console.
2. In the Topology view, select your virtual host (such as default\_host).
3. In the main pane, click the **Advanced** tab.
4. In the Host Aliases box, scroll down to the first empty line.
5. Click the line to get a cursor. Type the desired alias, and press Enter. Click the next line to add the next alias, as shown in Figure 3-42.

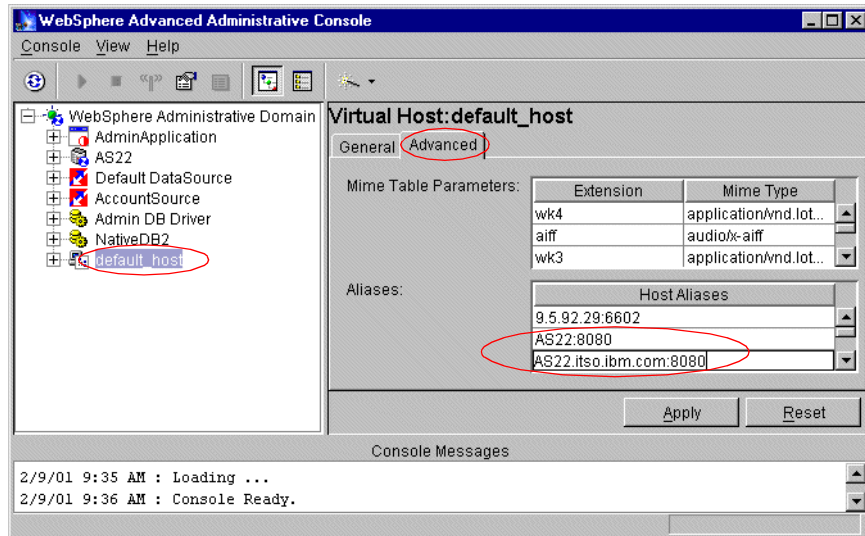


Figure 3-42 Defining a virtual host for WebSphere to recognize Domino calls

**Note:** The exact host name to be used in the URL must be present in the list for WebSphere to recognize it.

6. Click the **Apply** button.
7. From the Topology view, right-click your Application server to display the pop-up menu, then select **Restart** to restart your WebSphere Application Server to make the changes made above effective.

After completing these steps, you can forward requests to either the Domino HTTP server, using port 8080, or the iSeries HTTP server, by using the default port of 80. Using either HTTP stack you can also access WebSphere. Finally, you can still access Domino objects either through the iSeries HTTP server or through the Domino HTTP server.

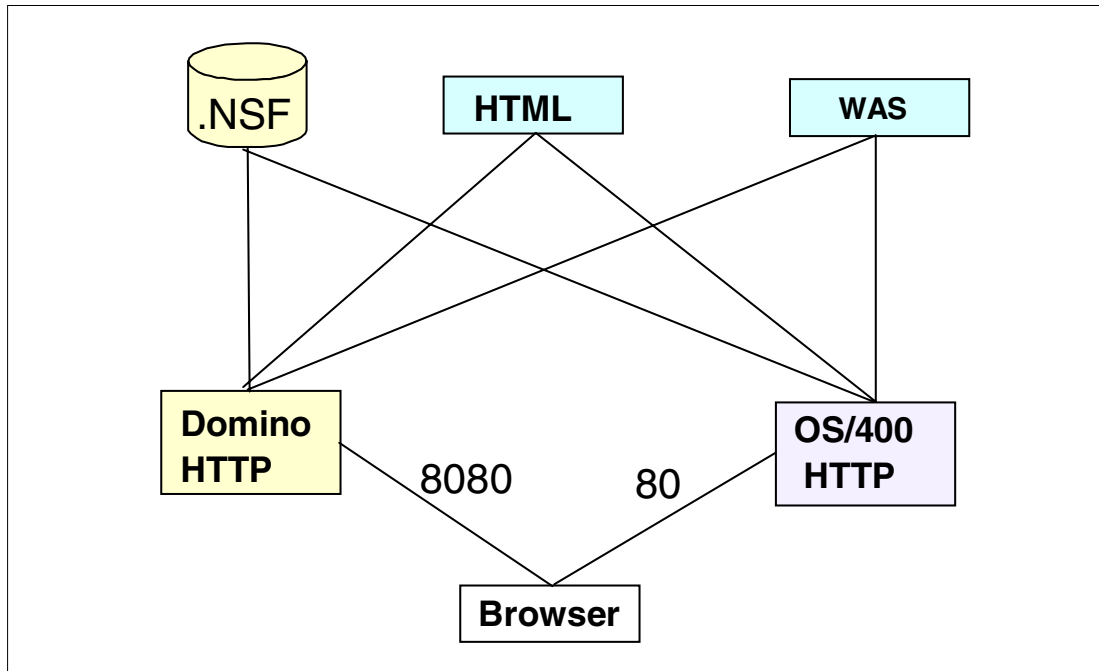
When the iSeries HTTP stack gets a request:

- ▶ HTML files are returned from the file system by iSeries HTTP server.
- ▶ Requests to WebSphere (for example, servlets) will be routed to the WebSphere for processing and be returned by the iSeries HTTP server.
- ▶ Requests for Domino objects will be routed to the Domino plug-in for processing and will be returned by the iSeries HTTP stack.

When the Domino HTTP stack gets a request:

- ▶ Requests for Domino objects will be routed to the Domino server for processing.
- ▶ All other requests will be forwarded to WebSphere through the DSAPI plug-in for processing.
- ▶ WebSphere will handle requests intended for it (for example, for servlets) and return all other requests to the Domino server running on port 8080.

The topology for this architecture is shown in Figure 3-43.



*Figure 3-43 Using Domino HTTP stack and iSeries HTTP stack together*

While it is possible to run both HTTP server stacks on the same host, it is not common practice to do so. This architecture offers little benefit. With both HTTP servers able to support the Domino and WebSphere integrated environment, it is beneficial from both an administrative point of view and a developer's point of view to use a single HTTP stack.





## Directory sharing using LDAP

WebSphere and Domino have the capability to share a common directory for users and groups. This common directory can be used to authenticate users and determine their membership in a group.

Having an LDAP (Lightweight Directory Access Protocol)-compatible directory available somewhere is crucial for implementing any kind of security with WebSphere. LDAP is a network protocol that lets any application access directory information and perform authentication services, such as validating user names and passwords. The application has to validate the user's identity and implement some kind of access control based on that identity. Domino has its own directory as part of the product, while WebSphere has no built-in directory at all. Also, when implementing the single sign-on function between WebSphere and Domino, an LDAP directory is mandatory.

When integrating Domino and WebSphere on the iSeries, you basically have two choices for an LDAP directory, either to use the LDAP directory support in Domino or use the OS/400 directory services support for LDAP.

In this chapter, we look at the options for directory sharing when using LDAP in an integrated environment of Domino and WebSphere on the iSeries server. The main purpose of this chapter is the preparation that is required in order to enable single sign-on between Domino and WebSphere. The topics covered in this chapter include:

- ▶ "What is LDAP?" on page 58
- ▶ "Using OS/400 LDAP" on page 60
- ▶ "Configuring WebSphere and Domino to use OS/400 LDAP" on page 80
- ▶ "Using Domino LDAP" on page 94
- ▶ "Configuring WebSphere to use Domino LDAP" on page 101
- ▶ "Securing WebSphere resources" on page 109

## 4.1 What is LDAP?

Lightweight Directory Access Protocol (LDAP) is an Internet protocol that provides access to information in directories.

LDAP is the standard for Internet directories in heterogeneous networks. Regardless of the directory's implementation, LDAP provides applications with a consistent view and access methods for information in the directory. Initially LDAP directories focused on personal information such as names and addresses, but LDAP directories are now becoming the foundation for network operating systems and systems management.

LDAP does not describe the directory service itself; rather it defines how and what information can be accessed. However, when referring to a directory that can be accessed using LDAP, the directory is usually called an LDAP directory. Therefore, LDAP directories can be implemented in many different ways. IBM implements cross-platform LDAP directories using Universal Database (UDB) through SecureWay Directory and as an integrated function of Lotus Domino.

With LDAP directories, information is organized as a tree, called a Directory Information Tree (DIT). Information is stored in the directory as objects or directory entries. These objects are referenced by using a distinguished name (DN). The distinguished name (DN) can be comprised of:

**cn** Common Name  
**ou** Organizational Unit  
**o** Organization  
**c** Country

This follows the syntax rules that were defined by the X.500 standards. An example of an object's DN is "cn=Wendy Thomson, ou=ITSO, o=IBM, c=us".

Software servers supporting LDAP are referred to as LDAP servers. They provide services that fall into one of the following categories:

- ▶ An LDAP server may provide LDAP access to existing directories, such as X.500 directories, Lotus Domino Directories or Novell Directory Services (NDS). The LDAP server provides proxy or gateway services allowing LDAP as an alternative directory access protocol.
- ▶ An LDAP server may provide LDAP access to existing (non-directory) databases of information. In this case, the LDAP server provides a "directory view" of this information to LDAP clients without converting or moving existing data into a directory. An example would be an LDAP server that provides access to an Oracle database. An LDAP server may provide access to its own directory where the information is only available by LDAP.
- ▶ The LDAP server and its data repository are specialized in providing the directory services to LDAP users. An example would be the LDAP servers provided by IBM SecureWay for IBM servers and Windows NT.

In all three of these cases, the directory supported by the LDAP server is considered an LDAP directory and can be accessed by an LDAP client the same way.

An LDAP client is a software application that accesses an LDAP server using a TCP/IP connection. The client accesses the LDAP directory using an industry-standard API, such as the LDAP C API, and does not need to know how the LDAP server stores the information. A client may log on anonymously to an LDAP server, thus seeing only public information, or authenticate as a specific user, in which case information to which that particular user is allowed access is also made visible.

LDAP directories may be located on a single server or configured across multiple servers. LDAP servers can replicate information between servers, making information more accessible. Synchronization of LDAP directories with non-LDAP directories provides for a *meta-directory* and a consistent method for all directory access.

IBM products use LDAP for authenticating users in the network, accessing information about users, managing product configurations, and managing network hardware/software inventories.

LDAP directories are now provided by IBM on OS/390, OS/400, Linux, AIX and Windows NT. All of these directories are compatible with each other and use DB2 UDB for storing directory information.

**Note:** The intent of this chapter is not to provide you with an in-depth discussion of LDAP, but rather give you enough information to understand what LDAP is and to be able to configure either OS/400 LDAP or Domino LDAP for the purposes of enabling authentication and authorization for users of Domino and WebSphere applications and ultimately to enable single sign-on between these two products.

#### 4.1.1 Options for directory sharing using LDAP on the iSeries server

There are two basic approaches to establishing a common directory shared by Domino and WebSphere on the iSeries server. One is to use the Domino Directory as the common directory for the WebSphere and Domino server. The other is to use another LDAP compatible directory service as the common directory. For this approach, the directory service choices are preferably those explicitly supported by WebSphere and Domino.

Using the Domino Directory as the common WebSphere and Domino directory service is attractive for those applications or environments where Domino is already established as the application base. Here, WebSphere would be used to complement Domino, providing support for those parts of an overall Domino application that require handling large numbers of transactions or guaranteed data integrity.

Current Domino Directory support allows Web users (as opposed to those using Lotus Notes clients to access the Domino server) and groups to be defined in the directory, along with the credentials necessary for authenticating Web access (such as the Internet password). The fact that the Domino Directory allows access via LDAP enables it to be easily established as the WebSphere user registry. In fact the options of "Domino 5.0" and "Domino 4.6" are listed as choices for the Directory Type field on the WebSphere Administrative Console window where the user registry is specified.

As previously mentioned, the WebSphere product supports the use of any LDAP-accessible directory as its user registry. Domino also supports the use of a separate LDAP directory for Web user authentication and access control in conjunction with the Domino Directory. It is possible then to set up a separate LDAP directory server and configure both WebSphere and Domino to use it as the user registry. Doing so allows the users requiring access to a combined WebSphere and Domino server to be managed in a single directory.

Using a separate LDAP directory would be a good choice for application environments where WebSphere is used as the main application engine and Domino is used for a subset of functions. For example, a virtual banking application could be supported mostly by HTML, servlets, JSPs, and EJBs served by WebSphere, with Domino providing a loan approval workflow function.

In this chapter we investigate both the IBM SecureWay Directory for OS/400 (or OS/400 LDAP) and the Domino Directory as choices for a common directory between WebSphere and Domino applications running on an iSeries server. We also cover how to configure both WebSphere and Domino to use either the IBM SecureWay Directory for OS/400 or the Domino Directory as a common directory for user authentication and access control. We then show you how to enable single sign-on between the two products in Chapter 5, “Single sign-on” on page 123.

## 4.2 Using OS/400 LDAP

One aspect of the Domino and WebSphere security models, the directory (referred to as the “user registry” in WebSphere), can be made common. This is possible since both Domino and WebSphere support the Lightweight Directory Access Protocol (LDAP) for directory access. The OS/400 Directory Services provides for LDAP access and Domino and WebSphere can be set up to use an LDAP accessible directory as the directory or user registry.

**Note:** OS/400 Directory Services is part of the IBM SecureWay Directory family of products and services and is sometimes referred to as SecureWay Directory for OS/400.

In this section we discuss configuring and administering the OS/400 LDAP server. Section 4.3, “Configuring WebSphere and Domino to use OS/400 LDAP” on page 80 then covers how to set up WebSphere and Domino to use OS/400 LDAP as the common directory or user registry for Web user authentication and authorization in a combined WebSphere and Domino environment on the iSeries server.

OS/400 Directory services includes an LDAP server and a complete set of LDAP clients and utilities that have been ported by IBM to many platforms. The same code is used across these platforms to insure consistent function, such as security, replication and interoperability of the servers.

The OS/400 LDAP server uses DB2 UDB for storing the directory information. DB2 UDB provides a robust database for storing, saving and recovering directory data. All directory operations make use of the transactional capabilities of DB2 UDB for OS/400 for insuring accurate directory updates. Directory data is backed up during standard OS/400 database operations. The LDAP server uses Structured Query Language (SQL) to search, add, delete and modify directory entries. Since OS/400 LDAP servers use DB2 UDB for OS/400, the servers are able to support additional directory search functions in addition to the functions defined by the LDAP standards.

You configure the OS/400 LDAP server using OS/400 Operations Navigator graphical user interface (GUI) similar to other OS/400 TCP/IP servers. In addition to configuring the LDAP server, Operations Navigator can be used to manage replication of the directory to other LDAP servers, configure access control lists (ACLs) and to start/stop the LDAP server.

The OS/400 LDAP server also supports certificates for security. The LDAP server's certificates are managed using OS/400 certificate management utilities similar to other OS/400 servers.

The OS/400 LDAP client supports accessing any LDAP server from all OS/400 ILE programming languages: C, COBOL, and RPG. An LDAP client for Windows is included with Client Access for OS/400 and a Java client is included in the OS/400 support of Java Naming and Directory (JNDI). Command-line utilities are provided for accessing an LDAP server from Windows and OS/400. These utilities are compatible with LDAP utilities provided for other operating systems and allow a search, add, modify, and delete of directory information.



Included with the OS/400 Directory Services are utilities for publishing information about OS/400 users in the System Distribution Directory or SDD into an LDAP directory. This includes their names, e-mail addresses, telephone numbers and user IDs. The information in the directory can be accessed by any LDAP client application, such as a mail client (Lotus Notes, Netscape Communicator, Microsoft Exchange, and so forth).

OS/400 also supports storing network, hardware and software inventory in an LDAP directory. Netfinity for OS/400 publishes workstation and server inventory in an LDAP directory. This allows for a search of an LDAP directory for network configuration information such as a workstation's system board information or an OS/400 software configuration.

If user information is stored in the OS/400 System Distribution Directory (SDD), synchronizing the SDD is an easy way to get started. Additional directory entries can be added to the directory for one user or a group of users using command-line utilities provided with Directory Services. To extend the management of user information, utilities can be developed or products purchased such as OS/400 Directory Assistant, which integrates with OS/400 Operations Navigator.

LDAP directory servers use a data definition called a schema to identify the type and format of information stored in the directory. IBM has developed a schema based on Internet standards and extensions required by IBM servers and products. A schema defines the objects and their properties that are stored in the directory. A directory object is defined by one or more object classes, such as ePerson. These object classes define a set of object properties called attributes to be used when accessing the directory object. For example, Directory Services uses several object classes (person, organizationalPerson, inetOrgPerson and ePerson) to define OS/400 users in the directory. Collectively, these object classes describe personal information such as a person's name, phone number, and e-mail address.

The IBM schema used by the Directory services LDAP server consists of several hundred object classes and over 600 attributes. The schema is defined in text files that can be extended and changed with a text editor as required by applications. For example, the ePerson object classes can be extended with additional attributes to store information specific to an organization. Once the schema is extended, LDAP-enabled applications can be developed to store additional information in the directory, thus allowing for management of all information about a user in one directory object. Once in the directory, applications can be developed to access the information from any platform in the enterprise.

An LDAP server can be configured on one iSeries server to support an LDAP directory in the network, and utilities provided with Directory Services enable user information to be published from other iSeries servers to this LDAP directory.

For more information on LDAP on the iSeries server, refer to the Web site:

<http://www.ibm.com/servers/eserver/iseries/ldap/>

## 4.2.1 Configuring OS/400 LDAP

Before you configure OS/400 LDAP, ensure the following software has been installed and tasks have been performed:

1. Verify that OS/400 Directory Services is installed. Prior to V5R1, Directory Services was a no charge option of OS/400 (option 32 of 5769-SS1), you just needed to make sure that it was installed. In OS/400 V5R1 or later, Directory Services is now a part of the base operating system.
2. Ensure OS/400 Operations Navigator is installed and configured. All directory server configuration tasks are performed using Operations Navigator.
3. An OS/400 user profile with \*ALLOBJ and \*IOSYSCFG special authorities is required.

4. Decide on a suffix (or naming context) for LDAP. This is a Distinguished Name (DN) that defines the name space for your directory. A Distinguished Name and password are the credentials a user offers to the server when signing in from the Web browser.

Suggestions for this DN include your organizational unit (ou), organization's name (o) and country (c). For example: ou=ITSO,o=IBM,c=us. Defining a suffix to the OS/400 LDAP server does not create a directory entry. A suffix simply identifies to the server that DNs in this namespace can be handled by the OS/400 LDAP server. If a user tries to sign on to your LDAP server with a different DN, it will result in a referral to another server or the server will return a "no such object" error.

5. Define an LDAP administrator DN and password. A client authenticated to the server using the LDAP administrator DN and password can create, delete, modify, and read all data in the directory.
6. Verify the local relational database directory LDAP will use for storage.
  - a. In V5R1, an LDAP server and a relational database directory are automatically created.
  - b. When using OS/400 V4R5 Operations Navigator or later with a V4R5 OS/400 or later, a new user library, QUSRDIRDB.LIB, is automatically configured as the LDAP database library.
7. Check the OS/400 system value of QALWUSRDMN (Allow user domain objects in libraries) with the Work with System Values (WRKSYSVAL) command. If you have previously changed the QALWUSRDMN system value from \*ALL, make sure the system library, QDIRSRV2 is included as a value. Otherwise you will not be able to publish information from the System Distribution Directory (SDD) to the LDAP directory.

You are now ready to configure the OS/400 LDAP server. The following steps assume you are configuring the LDAP server for the first time. If you are changing a configuration, be careful to consider existing settings. Perform the following steps:

1. Launch OS/400 Operations Navigator and open the TCP/IP Servers folder. This is accessed by selecting **Network -> Servers -> TCP/IP** in the Operations Navigator tree (see Figure 4-1).

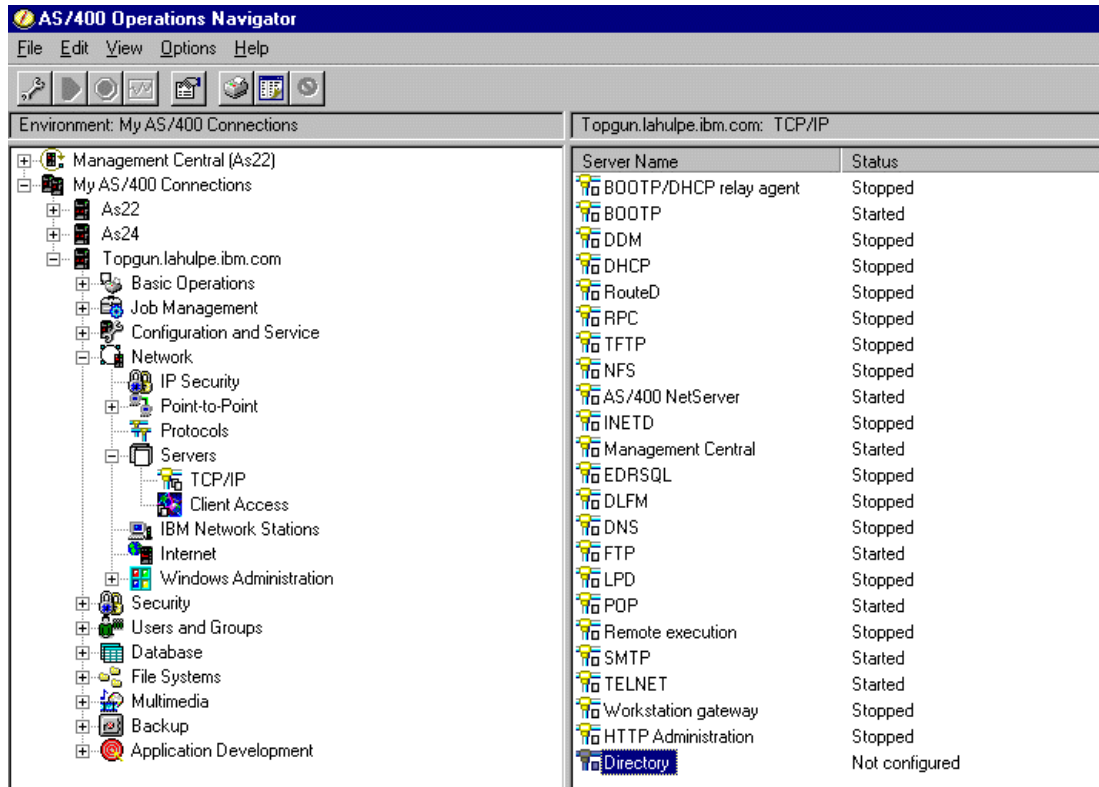


Figure 4-1 Directory option in Operations Navigator

2. Right-click the **Directory** option that appears in the TCP/IP pane on the right-hand side of the Operations Navigator window. This brings up a pop-up context menu from which you should select the **Configure** option (see Figure 4-2).

**Note:** If you are using OS/400 V5R1 or have already previously attempted configuring the Directory Server, you must first stop the Directory Server and select the **Reconfigure** drop-down option instead.

If you are simply adding a new Directory suffix, you should choose the **Properties** option.

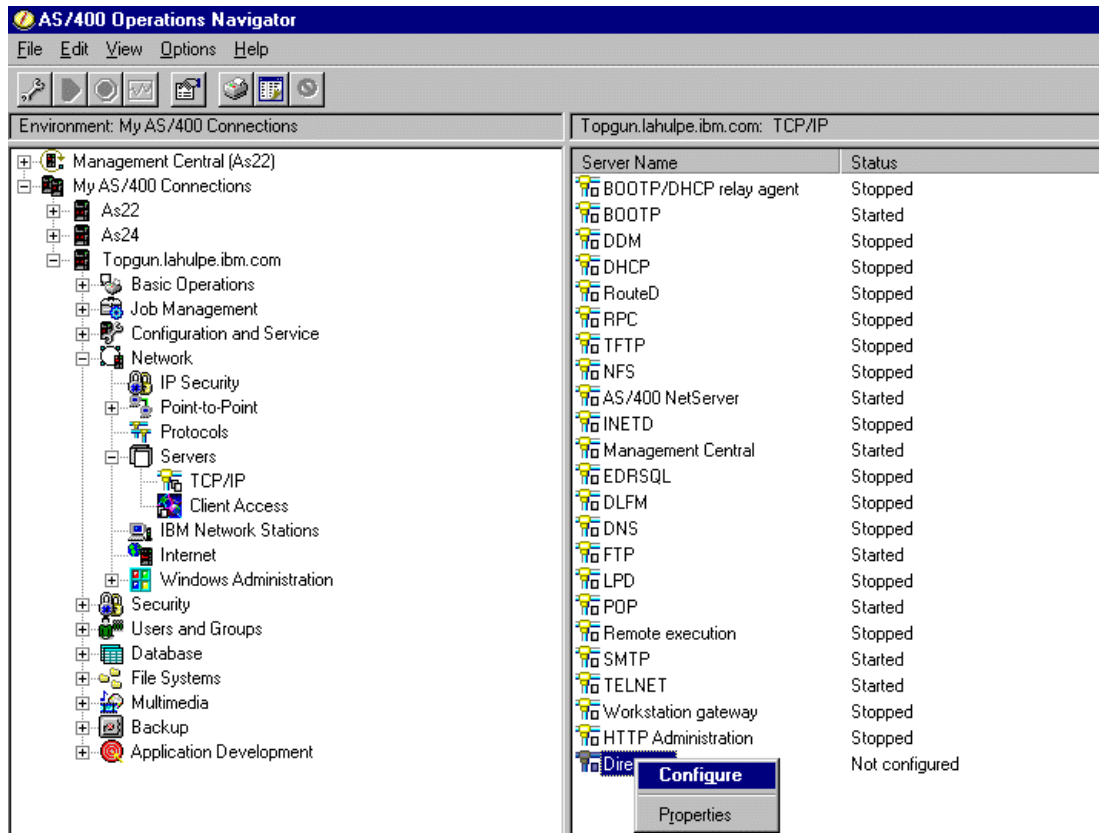


Figure 4-2 Directory configuration option drop-down in Operations Navigator

3. If you have already attempted to configure the Directory Server, you will need to select the **Reconfigure** option instead. See Figure 4-3.

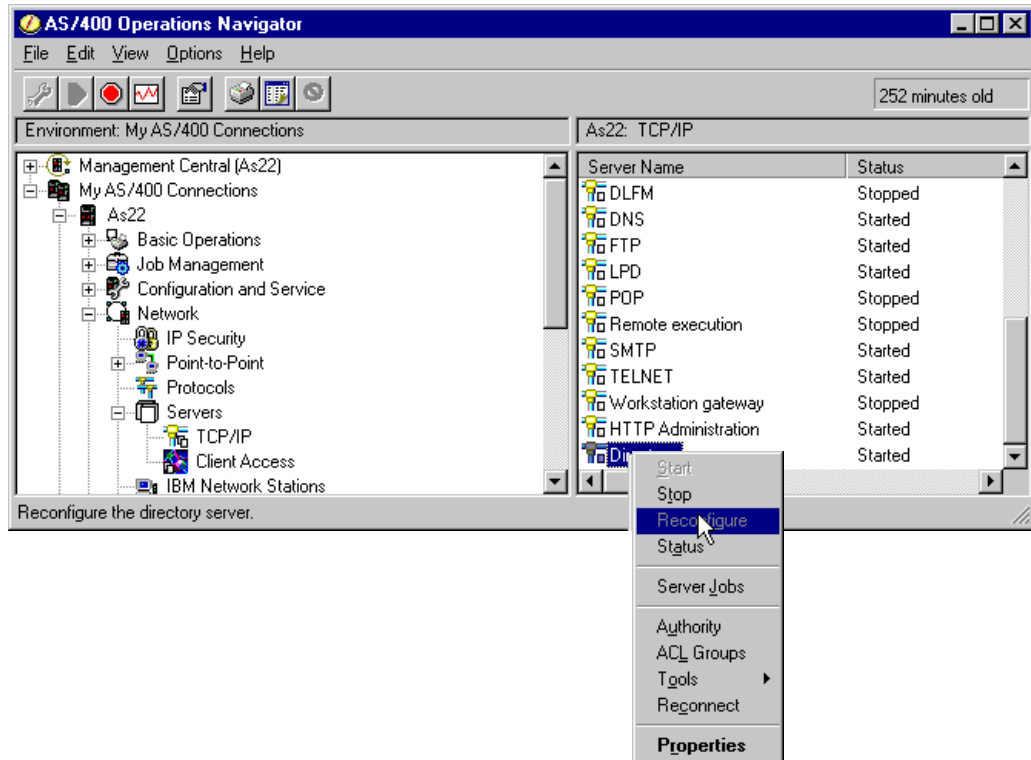


Figure 4-3 Reconfigure option for the Directory Server

4. The Configure Directory Server wizard window appears (see Figure 4-4). Click **Next**.

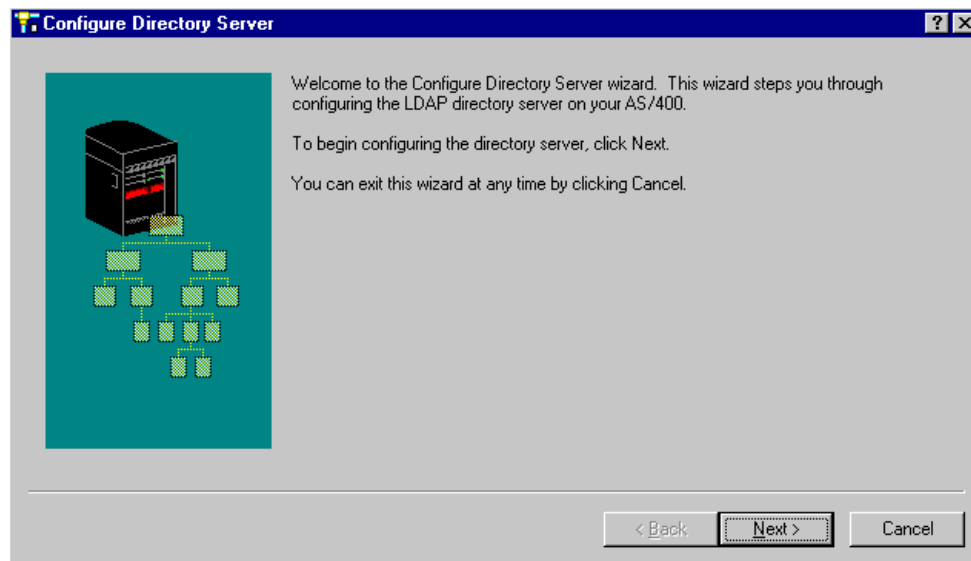
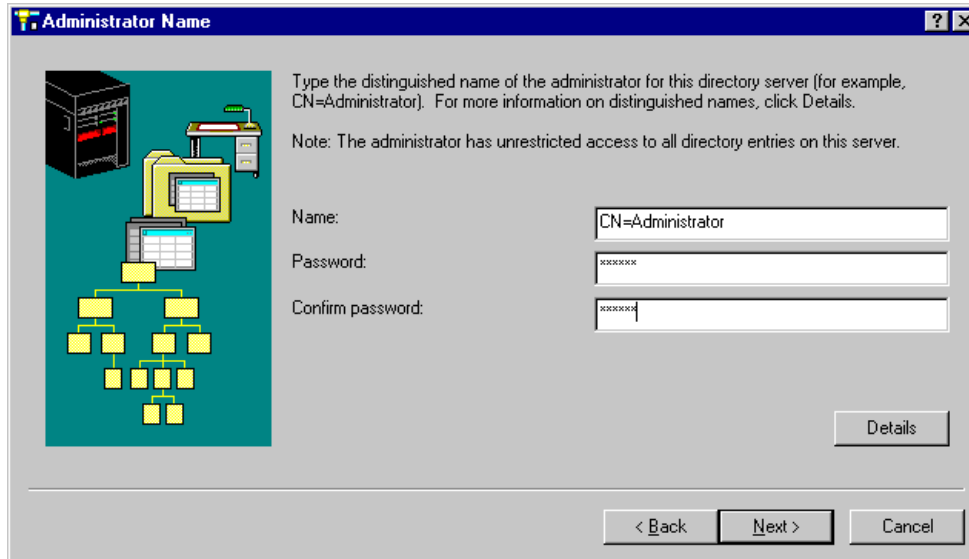


Figure 4-4 Configure Directory Server wizard

5. On the Administrator Name window, enter the name and password for the LDAP administrator. The default name is cn=Administrator, but this can be changed. For this example it is left as the default. Click **Next** (see Figure 4-5).



**Administrator Name**

Type the distinguished name of the administrator for this directory server (for example, CN=Administrator). For more information on distinguished names, click Details.

Note: The administrator has unrestricted access to all directory entries on this server.

Name:

Password:

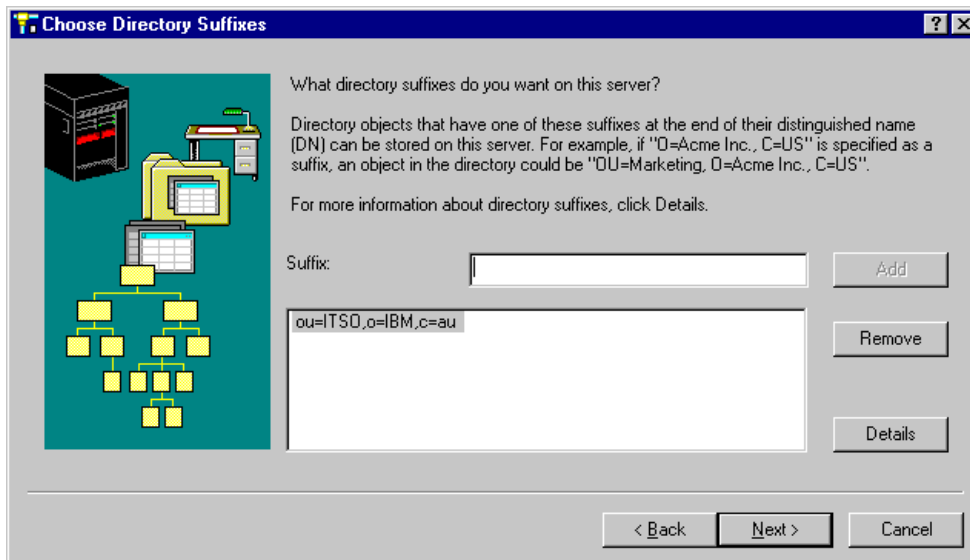
Confirm password:

[Details](#)

[< Back](#) [Next >](#) [Cancel](#)

Figure 4-5 LDAP Administrator name and password

- On the Choose Directory Suffixes window, the directory suffix needs to be added to the Directory server. Specify the lowest level of the hierarchy first. In this example we have started at the Organizational Unit (ou) of ITSO, then the Organization (o) of IBM, then the Country (c) of us. Click **Add** then click **Next** to continue (see Figure 4-6).



**Choose Directory Suffixes**

What directory suffixes do you want on this server?

Directory objects that have one of these suffixes at the end of their distinguished name (DN) can be stored on this server. For example, if "O=Acme Inc., C=US" is specified as a suffix, an object in the directory could be "OU=Marketing, O=Acme Inc., C=US".

For more information about directory suffixes, click Details.

Suffix:

[Add](#)

[Remove](#)

[Details](#)

[< Back](#) [Next >](#) [Cancel](#)

Figure 4-6 Add directory suffix

- On the Start Server when TCP/IP is Started window, the **Yes, start this server when TCP/IP is started** option is already checked. Uncheck this box only if you do not want LDAP to start with TCP/IP. Click **Next** (see Figure 4-7).

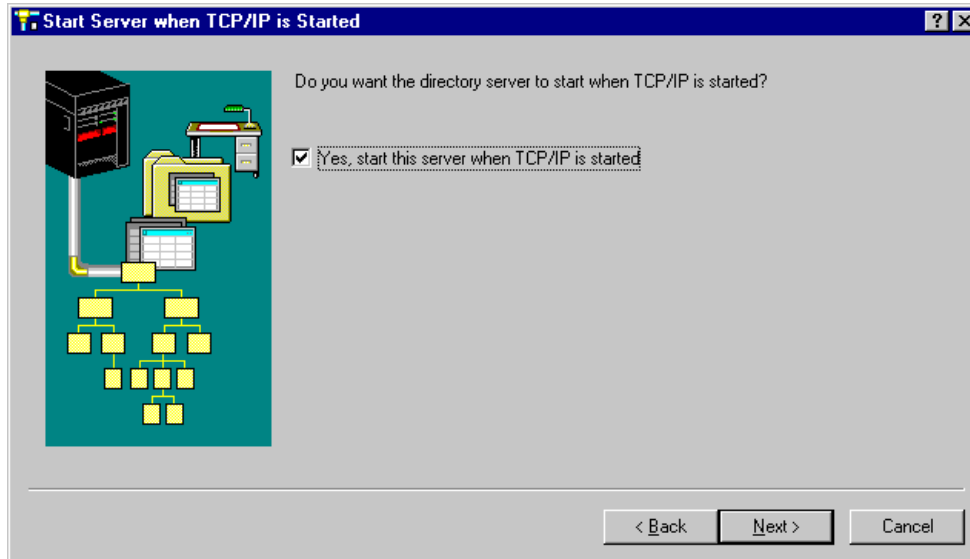


Figure 4-7 Start LDAP server when TCP/IP is started option

8. On the final Configuration Summary window, check to make sure the information is correct, then click **Finish** and your Directory server is configured. You can now start the Directory server (see Figure 4-8).

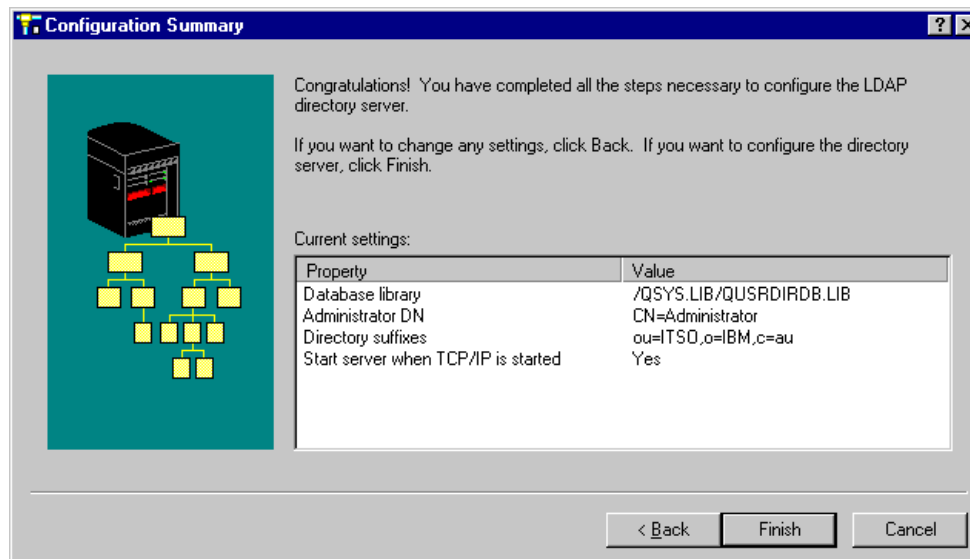


Figure 4-8 Configuration summary

## 4.2.2 Starting and stopping the OS/400 LDAP server

The OS/400 LDAP server can be started in a number of different ways:

- ▶ When configuring or re-configuring the LDAP server using the Operations Navigator Configure Directory Server wizard, there is a check box option to start your Directory Server automatically whenever TCP/IP is started. If you checked that option, your LDAP server is automatically started whenever TCP/IP is started on your iSeries server.
- ▶ You can also specify an automatic start for your Directory server, and any other TCP/IP server, by using the Servers to Start option on the TCP/IP properties window in

Operations Navigator. This is accessed by right-clicking **Network -> Protocols -> TCP/IP** in Operations Navigator and selecting **Properties** from the pop-up context menu).

- ▶ Manually from Operations Navigator, click **Network -> Servers -> TCP/IP** and right-click the Directory entry (see Figure 4-3 on page 65) and select the **Start** option from the pop-up context menu.
- ▶ Manually using the OS/400 CL command of Start TCP/IP Server (STRTCPSVR):  
`STRTCPSVR SERVER(*DIRSRV)`

The OS/400 LDAP server can be stopped in two ways:

- ▶ Manually using the OS/400 CL command of End TCP/IP Server (ENDTCPSVR):  
`ENDTCPSVR SERVER(*DIRSRV)`
- ▶ From Operations Navigator, click **Network -> Servers -> TCP/IP** and right-click the Directory entry (see Figure 4-3 on page 65) and select the **Stop** option from the pop-up context menu.

**Note:** The \*DIRSRV variable in the STRTCPSVR or END TCPSVR command tells OS/400 to start or stop the OS/400 Directory Server. These commands can also be run from within CL programs.

### 4.2.3 Publishing OS/400 SDD entries to the LDAP directory

Once you have configured and started the OS/400 LDAP server on your iSeries, you have the capability to publish your iSeries users in the OS/400 System Distribution Directory or SDD to the LDAP directory. Once the users are published to the LDAP directory, the SDD can be automatically synchronized with the LDAP directory.

Two agents are predefined by OS/400 Directory Services for you to use. These agents are called Users and Computers. The User's agent will publish users from the OS/400 System Distribution Directory (SDD) to an LDAP server. The Computer's agent will publish information about your iSeries hardware and software.

The Users agent publishes users from the SDD to an LDAP server and keeps the LDAP directory synchronized with changes made in the SDD. You can then use the information that you publish in the LDAP directory from other LDAP applications that access address book information. Changes made in the LDAP directory are not published back to the SDD. The types of users that are published are local users and remote users with an SMTP address. Shadowed users and remote users that do not have a SMTP address are not published. Publishing is done to one LDAP server and one directory path.

The Computer's agent publishes information about your iSeries to the LDAP directory. Directory Services will publish limited information identifying your iSeries server and the operating system.

#### SDD to LDAP mapping

LDAP uses the distinguished name (DN) as the unique name for an entry. For the SDD entries in the LDAP directory, the DN is the common name (cn) combined with the directory path that is configured. OS/400 supports LDAP Version 3. The SDD entry is exported to the LDAP directory by using the ePerson object class in OS/400 V4R5. Table 4-1 describes the mapping of SDD fields to attributes of the ePerson object class.



Table 4-1 Mapping of SDD fields to the ePerson object class

System Distribution Directory field	LDAP attributes for ePerson object class
User profile	uid
Description	description
Last name	sn(surname), cn (common name)
First name	givenName, cn (common name)
Preferred name	cn (common name)
Full name	cn (common name)
User ID	cn (common name)
Department	department number
Job title	title
Telephone number 1 & 2	telephoneNumber
FAX telephone number	facsimileTelephoneNumber
Office	roomNumber
Address lines 1-4	registered Address
SMTP name	mail

The common name uses the following formats:

- ▶ 'First name' 'Middle Name' 'Last name'
- ▶ 'Preferred name' 'Last name'
- ▶ 'Full name'
- ▶ 'UserID'

So for example, a user with the first name of 'Jonathan', preferred name of 'John', middle name of 'T.', last name of 'Smith' and user ID of 'JSMITH', would have the common names of:

- ▶ cn=Jonathan T. Smith
- ▶ cn=John Smith
- ▶ cn=Smith, Jonathan T. (John)
- ▶ cn=JSMITH

The distinguished name is the first common name (cn) combined with the directory path. So for example, if the directory path is 'ou=chicago, o=acme, c=us', the distinguished name (DN) for this user would be 'cn=Jonathan T. Smith,ou=chicago, o=acme, c=us'.

If you have two users in the SDD that resolve to the same DN, they overlay each other in the LDAP directory. Sometimes overlaying names is what you want if you are merging multiple OS/400 SDDs into one LDAP directory. If you have different users with the same name, ensure they have different distinguished names to prevent overlaying each other.

## Publishing SDD users

To publish your System Distribution Directory users to the LDAP directory, perform the following steps:

1. Verify TCP/IP is configured on your system. Enter the Change TCP/IP Domain CL command (CHGTCPDMN) from the OS/400 command line, press F4, and ensure the host and domain name is set, then press F3 to exit.

2. Verify the SMTP information is configured. Enter the Change SMTP Attributes CL command (CHGSMTPA) from the OS/400 command line, press F4, and verify the user ID delimiter. You must press Enter, because this sets the SMTP default information that may be needed for publishing the mail information to LDAP if the user does not have SMTP information in their SDD entry.
3. You should check the system value of QALWUSRDMN (Allow user domain objects in libraries) by using the Work with System Values (WRKSYSVAL) command. If you have changed the QALWUSRDMN system value from \*ALL, make sure that it includes QDIRSRV2. You will not be able to publish SDD information otherwise.
4. From OS/400 Operations Navigator, there is an initial display of iSeries server names. Right-click the system name that you want to publish the SDD from and select **Properties** (see Figure 4-9).

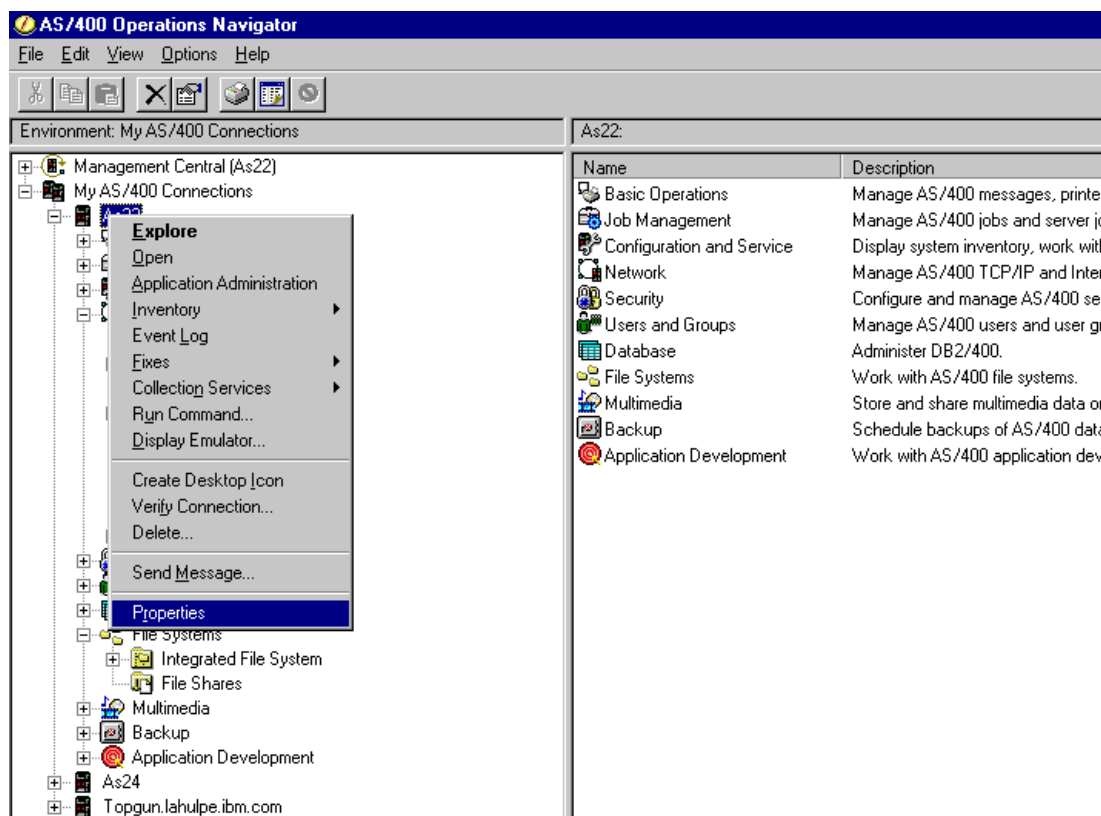


Figure 4-9 Operations Navigator iSeries server properties

5. From the Properties window, click the **Directory Services** tab. Click **Users** from the list that is displayed to highlight it and then click the **Configure** button (see Figure 4-10).

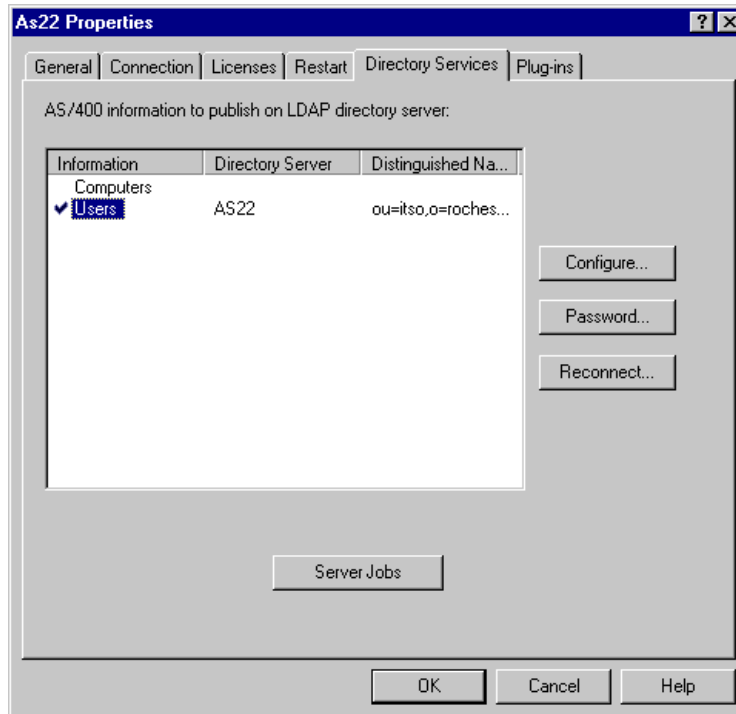


Figure 4-10 Directory services tab

6. The Directory Services Publishing - Configure window appears. Select the **Publish AS/400 information for** check box. Enter the following values for the remaining fields:
  - In the Where to Publish section, for the Directory server parameter, specify the iSeries server name that the LDAP server is running on. In our example this is ASXX.
  - For the Under DN parameter, enter the DN name you specified when you configured the LDAP server. In our example it is ou=itso,o=rochester,c=us.
  - In the Server Connection section, specify the Distinguished Name of the LDAP server administrator. In our example this is cn=Administrator. This was created when you configured the LDAP server.
  - Enter the LDAP Administrator's password that you specified when you configured the LDAP server.
  - You only need to specify using SSL if required.
  - Leave the port setting at 389, since this is the default for the LDAP server, unless you changed this in the LDAP server configuration.

See Figure 4-11.

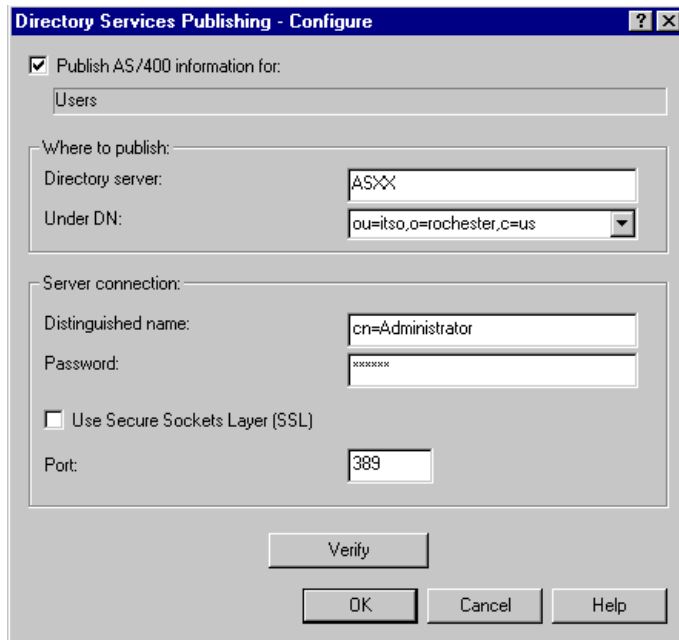


Figure 4-11 Configure connection for publishing SDD

7. Click the **Verify** button to make sure that the directory path you specified exists on the LDAP server.

**Tip:** Be sure to click **Verify**. In addition to verifying that the LDAP server and connection information is correct, it also verifies that the parent DN exists, and if not, optionally creates it for you.

8. If the directory path does not exist, you are prompted to create the path. If you do not create the path, publishing is not successful. Click **Yes** (see Figure 4-12).

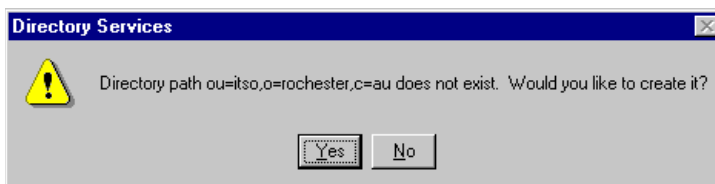


Figure 4-12 Create directory path prompt

9. Once the Directory Services settings verified successfully message is received (see Figure 4-13), click **OK** and then click **OK** again on the Directory Services Publishing - Configure window. The SDD will be synchronized with the LDAP directory every five minutes.

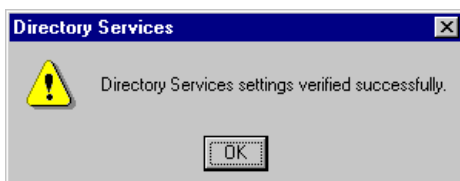


Figure 4-13 Directory Services settings verified successfully message

You now have the method for publishing OS/400 system distribution directory entries to LDAP and keeping changes that you make to a system distribution directory user synchronized with the LDAP directory you publish to.

You can publish SDD users to an LDAP directory that is not on an iSeries server. The LDAP server needs to have the inetOrgPerson, Publisher, and ePerson object classes defined in the schema file on that server. These object classes are used in LDAP to store the SDD information. Documentation on the inetOrgPerson and Publisher object classes can be found in the iSeries Information Center and information on ePerson can be found in the IBM SecureWay Directory. You can also find the object classes in the LDAP schema files on the iSeries server in the directory /QIBM/ProdData/OS400/DirSrv.

OS/400 Directory Services provides additional support for storing and retrieving information in an LDAP directory. A set of publishing APIs is provided that allows an application to send data to an LDAP server without the application needing to manage such information as the server address, how to connect to the server, and where in the directory to put the information. These APIs will also retry operations that fail due to errors such as the server being down. The publishing APIs are documented in the iSeries Information Center in the Directory Services topic.

To use these APIs, you first configure a publishing agent with the information necessary to connect and authenticate to a server (such as server, port, bind DN) and the location of your data in the directory (parent DN). The remaining APIs are used to publish objects under one of these agents. From OS/400 Operations Navigator, you can modify the publishing agent configuration from the Directory Services property page of the system you wish to configure.

## Preventing select SDD users from being published

You can prevent specific users from being published, although some entries are prevented automatically from being published to LDAP. They are the \*ANY SDD entries and some other IBM-supplied entries starting with Q (QSECOFR, QDOC, QSYS, QDFTOWN, QUSER for example). A specific user can be prevented from being published to LDAP by performing the following steps:

1. Add the user-defined field QREPL QLDAP to the system distribution directory. This only needs to be done once per system. Enter the following OS/400 CL command:  

```
CHGSYSDIRA USRDFNFLD((QREPL QLDAP *ADD *DATA 4))
```
2. Specify \*NO as the value for the QREPL QLDAP user-defined field for those users that you do not want to replicate to LDAP. Any other value or absence of the QREPL QLDAP user-defined field will replicate the user. We recommend that you either leave the QREPL QLDAP value blank or specify \*YES if you want the user to be replicated.

For example, using the Work with Directory Entries (WRKDIRE) CL command, option 1 to add a user or option 2 to change a user, press the F20 key to specify user-defined fields. When using the ADDDIRE or CHGDIRE commands, specify the USRDFNFLD((QREPL QLDAP \*NO)) parameter to prevent the user from being replicated. If the user is already replicated to LDAP and \*NO is specified in the QREPL QLDAP user-defined field, the user will be deleted from the LDAP directory. Likewise, if the value of the QREPL QLDAP user-defined field is changed to anything but \*NO, the user will be added to the LDAP directory.

## 4.2.4 Verifying the connection to the OS/400 LDAP server

Once you have configured and started the OS/400 LDAP server on your iSeries server or when you have published OS/400 System Distribution Directory (SDD) users to the LDAP directory, you should check the connection to the OS/400 LDAP directory and verify the SDD entries were published. There are several different methods for doing this. In this section we explain how to use a Web browser and the QShell utility.

### Using a Web browser

To verify a connection to the LDAP server from your Web browser, perform the following steps:

1. Open Netscape Navigator and specify the location of your LDAP server. For example:

```
ldap://SYS400.IBM.COM
```

Here SYS400.IBM.COM is the full domain name of your iSeries server.

2. You should see a window full of information about your LDAP server.
3. To verify your OS/400 user ID has been published from the SDD to the LDAP directory, enter the following information in the Netscape location field:

```
ldap://SYS400.IBM.COM/cn=Joe Admin,ou=itso,o=rochester,c=us
```

Here SYS400.IBM.COM, Joe Admin, and ou=itso,o=rochester,c=us are entries from your own environment.

**Note:** You cannot verify a connection with Internet Explorer or certain newer versions of Netscape Navigator.

4. You should now see a window that displays the user ID information.

### Using the Qshell utility

To verify a connection to the LDAP server using the Qshell utility, perform the following steps:

1. Open a 5250 session to your iSeries server and start the Qshell Interpreter by entering the following OS/400 command:

```
STRQSH or QSH
```

2. To search and display all directory entries, enter the following command:

```
ldapsearch -v -D cn=Administrator -w ldappw -b ou=itso,o=rochester,c=us  
"(objectclass=*)"
```

Where ldappw is the LDAP Administrator's password. Also, note the command may wrap around the command line.

3. To check the LDAP Administrator password is correct, enter the following command:

```
ldapsearch -v -D cn=Administrator -w ldappw -b cn=monitor -s base "(objectclass=*)"
```

Where ldappw is the LDAP Administrator's password.

For more information on using the Qshell Interpreter on OS/400, see "OS/400 Qshell utilities (QSH)" on page 75.

## 4.2.5 Managing directory entries

There are several different ways to manage the OS/400 LDAP directory entries. You can use the IBM SecureWay Directory Management Tool, which is the PC client, or you can use the OS/400 Qshell Interpreter from a 5250 session. Alternatively, you could develop your own application using the C LDAP APIs that are provided as part of OS/400 Directory Services.

### IBM SecureWay Directory Management Tool (DMT)

The IBM SecureWay Directory Management Tool (DMT) provides you with a graphical user interface for managing LDAP directory content. The tasks that you can perform with the DMT include the following:

- ▶ Browsing directory schema
- ▶ Adding, editing, and deleting object classes
- ▶ Adding, editing, and deleting attributes
- ▶ Browsing and searching the directory tree
- ▶ Adding, editing, viewing, and deleting entries
- ▶ Editing entry Relative Distinguished Name (RDNs)

The DMT is part of the Windows LDAP client that is included with OS/400 Directory Services. The client is shipped in an OS/400 Integrated File System (IFS) directory. DMT can only be used with IBM SecureWay Directories that support LDAP V3 (OS/400 V4R5 or later).

Perform the following steps to install the Windows LDAP client, including the DMT, onto a PC workstation:

1. In Operations Navigator, expand **File Systems**.
2. Expand **File Shares**.
3. Double-click **Qdircrv**.
4. Double-click **UserTools**.
5. Double-click **Windows**.
6. Double-click **setup.exe** to start installing the DMT.
7. Follow the on-screen instructions to complete the installation.

Documentation for the IBM SecureWay Directory Management Tool (DMT) is located in the file dparent.htm. This file is copied to the IBM SecureWay Directory folder on your PC workstation when you install the LDAP client.

### OS/400 Qshell utilities (QSH)

OS/400 Directory Services includes five utilities that allow you to perform actions on the LDAP directory server from the Qshell Interpreter on OS/400. These utilities use the LDAP APIs. You can use these utilities from the Qshell command line or call them from your programs. You may also find them useful as programming examples. When you install the Windows LDAP client that is included with OS/400 Directory Services, you also install code that is very similar to the source code for the Qshell utilities.

The Qshell utilities are as follows:

- ▶ **ldapmodify** and **ldapadd** utilities, which add and modify LDAP directory entries. The **ldapmodify** utility allows you to change entries or add entries to the LDAP directory server from the QSH command shell on OS/400. It uses the **ldap\_modify** application program interface (API). The **ldapadd** utility, which uses the **ldap\_add** API, works identically to the **ldapmodify** utility with the exception that the **-a** flag is turned on automatically.
- ▶ **ldapdelete** utility, which removes entries from the LDAP directory.
- ▶ **ldapsearch** utility, which searches the LDAP directory for entries.

- `ldapmodrdn` utility, which modifies the Relative Distinguished Name (RDN) of LDAP directory entries.

To access the Qshell Interpreter, use the OS/400 CL command, `Start QSH` (QSH or STRQSH).

### **Qshell syntax format**

The following is the Qshell syntax to modify and add entries in the LDAP directory:

```
ldapmodify [-a] [-V] [-b] [-c] [-r] [-n] [-v] [-F] [-R] [-C charset] [-d debuglevel] [-D binddn] [-w passwd] [-h ldaphost] [-p ldapport] [-f file] [-Z] [-K keyfile] [-P keyfilepw] [-N certificatename]
```

```
ldapadd [-V] [-b] [-c] [-r] [-n] [-v] [-F] [-R] [-C charset] [-d debuglevel] [-D binddn] [-w passwd] [-h ldaphost] [-p ldapport] [-f file] [-Z] [-K keyfile] [-P keyfilepw] [-N certificatename]
```

Table 4-2 explains all the LDAP Qshell parameters.

*Table 4-2 Qshell syntax*

Parameters	
-V	Specifies the LDAP version that the utility uses to bind to the LDAP server. By default, it uses an LDAP V3 connection. To explicitly select LDAP V3, specify "-V 3". Specify "-V 2" to run as an LDAP V2 application.
-a	Only <code>ldapmodify</code> uses this parameter. It indicates that the utility will add entries rather than modifying them. Using this parameter is the same as using <code>ldapadd</code> .
-b	Assume that any values that start with a '/' are binary values and that the actual value is in a file whose path is specified in the place where values normally appear.
-c	Continuous operation mode. Errors are reported, but <code>ldapmodify</code> or <code>ldapadd</code> continues with modifications or adds. The default is to exit after an error.
-r	Replace existing values by default.
-n	Show what would be done, but do not actually modify entries. Useful for debugging in conjunction with -v.
-v	Use verbose mode, with many diagnostics written to standard output.
-F	Force application of all changes regardless of the contents of input lines that begin with replica: (by default, replica: lines are compared against the LDAP server host and port in use to decide if a replication log record should actually be applied).
-R	Specifies that referrals are not to be automatically followed.
-C <i>charset</i>	Specifies that strings supplied as input to the utility are represented in a local character set ( <i>charset</i> ), and must be converted to UTF-8. Use the -C <i>charset</i> option if the input string codepage is different from the job code page value. Refer to the documentation for the <code>ldap set iconv charset()</code> API to see supported <i>charset</i> values
-d <i>debug level</i>	Sets the debug level to <i>debuglevel</i>
-D <i>binddn</i>	Use <i>binddn</i> to bind to the LDAP directory, <i>binddn</i> should be a string-represented DN.
-w <i>passwd</i>	Use <i>passwd</i> as the password for simple authentication.
-h <i>ldaphost</i>	Specify an alternate host on which the LDAP server is running.



Parameters	
<b>-p</b> <i>ldapport</i>	Specify an alternate Transmission Control Protocol (TCP) port where the LDAP server is listening. The default LDAP port is 389. If not specified and <b>-Z</b> is specified, the default ZLDAP SSL port 636 is used.
<b>-f</b> <i>file</i>	Read the entry modification information from an LDIF file instead of from standard input. If an LDIF file is not specified, you must use the standard input to specify the update records in LDIF format.
<b>-Z</b>	Use a secure SSL connection to communicate with the LDAP server. The <b>-Z</b> option is only supported by SSL-enabled versions of this tool.
<b>-K</b> <i>keyfile</i>	Specify the name of the SSL key database file. If the key database file is not in the current directory, specify the fully-qualified key database filename. If the utility cannot locate a key database, it will use a "hard-coded" set of default trusted certificate authority roots. The key database file typically contains one or more certificates of certification authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. This parameter effectively enables the <b>-Z</b> switch.
<b>-P</b> <i>keyfilepw</i>	Specify the key database password. This password is required to access the encrypted information in the key database file (including the private key). If a password stash file is associated with the key database file, the password is obtained from the stash file and this parameter is not required. This parameter is ignored if either <b>-Z</b> nor <b>-K</b> are specified
<b>-N</b> <i>certificatename</i>	Specify the label associated with the client certificate in the key database file. Note that if the LDAP server is configured to perform Server Authentication only, a client certificate is not required. If the LDAP is configured to perform Client and Server Authentication, a client certificate is required. <i>certificatename</i> is not required if a default certificate/private key pair has been designated as the default. Similarly, <i>certificatename</i> is not required if there is a single certificate/private key pair in the designated key database file. This parameter is ignored if neither <b>-Z</b> nor <b>B</b> are specified.

You can find more information on the LDAP Qshell utilities in the iSeries Information Center by clicking **Networking -> TCP/IP -> TCP/IP Services and Applications -> Directory Services (LDAP) -> LDAP command line utilities**.

## C LDAP APIs

You can also develop your own application using the C LDAP APIs that are provided as part of OS/400 Directory Services. The C APIs are widely portable, based on a draft Internet standard.

Documentation for these APIs is located in iSeries Information Center. Click **Programming -> CL and APIs -> iSeries APIs -> APIs by category**. Select the **Directory Services** category.

## LDAP Data Interchange Format (LDIF)

The LDAP standards (RFC 2849) define a standard data interchange format. This defines a format for a text file that can be used to import data into the LDAP directory. For example, to add a new directory entry, you could create an LDIF containing the following lines:

```
dn: cn=Wendy Thomson, ou=ITS0, o=IBM, c=us
objectclass: ePerson
cn: Wendy Thomson
sn: Thomson
givenName: Wendy
mail: wthom@ibm.com
telephoneNumber: 555.555.5555
```

ou: ITS0  
uid: WENDYT

You can write an application that creates such a file from an existing database or creates a file containing periodic updates. For more information on LDIF files, refer to the iSeries Information Center Web site at:

<http://publib.boulder.ibm.com/pubs/html/as400/infocenter.htm>

### ***Importing and exporting LDIF***

You can also import an LDIF file into the directory using the OS/400 QgldImportLdif APIs or by using Operations Navigator. In Operations Navigator, IBM provides tools to import and export LDIF files from the OS/400 LDAP Directory. These tools can be found in the Directory drop-down context menu (accessed by right-clicking **Network -> Servers -> TCP/IP -> Directory**; see Figure 4-3 on page 65) and selecting the **Tools** option from that menu. The LDIF import utility can be used only to create new entries. It will not update existing entries.

## **4.2.6 Using LDAP for authentication**

Using LDAP for authentication is straightforward. You simply use LDAP APIs to connect to an LDAP server using credentials provided by the user. If the authentication (bind) operation is successful, the user is considered authenticated.

The credentials provided by the user could take the form of an LDAP DN and password. Or, rather than providing a DN, the application may be given the user's name or user ID. The application can perform an LDAP search to find the DN to use on the bind request. For example, given the user name Wendy Thomson, a search for cn=Wendy Thomson would provide the entry used in the examples. With the password the user entered, the application then binds as:

cn=Wendy Thomson,ou=itso,o=ibm,c=us

**Tip:** If you are creating an LDAP entry for a person that also has an OS/400 user profile, you can set up the entry so that it uses the user's OS/400 password. You do this by creating an entry with a "uid" attribute that contains the user profile name, but which does not have a userPassword attribute. In this case, the server verifies that the password used to authenticate is correct for the OS/400 user profile identified in the uid attribute. For our Wendy Thomson example, we can do this in the following way:

dn: cn=Wendy Thomson, ou=itso, o=ibm, c=us  
uid: WENDYT

## **4.2.7 Saving the LDAP directory and problem determination**

This section covers how to save the LDAP directory and some of the common problems you may encounter.

### **Saving the LDAP directory**

The LDAP directory can be backed up by saving the directory library. For OS/400 V4R5, when the server is first started, several database tables are created in the QUSRDIRDB database library. For OS/400 V4R3 and V4R4, you define the DB2/400 database library to use. To back up the directory contents, you'll want to save everything in this library.

## Viewing the LDAP server job log

In some cases you may want to view the LDAP server job log. To do this directly from Operations Navigator, select **Server Jobs** from the Directory server context menu. From the green screen, use the Work with Active Jobs (WRKACTJOB) command for the QDIRSRV job, as follows:

```
WRKACTJOB JOB(QDIRSRV)
```

This brings you to the Work with Job window where you can find messages such as failed bind attempts and various directory errors. The job log can be useful when debugging LDAP applications and the LDAP return code doesn't give you enough information to determine what is wrong.

## Common errors

The following are some common errors you could encounter when installing and configuring the LDAP server on the iSeries server.

### ***ldap\_bind: No such object***

A common cause of this error is that a user makes a typing mistake when performing an operation. Another common cause is when the LDAP client attempts to bind with a DN that does not exist. This often occurs when the user specifies what he or she mistakenly thinks is the administrator DN. For example, the user may specify QSECOFR or Administrator, when the actual administrator DN may be something like "cn=Administrator". For details about the error, look at the QDIRSRV job log.

### ***ldap\_bind: Inappropriate authentication***

This error is usually generated when the client attempts to bind with a password that is not valid. To obtain details about the error, look at the QDIRSRV job log.

### ***[Failing LDAP operation]: Insufficient access***

This error is usually generated when the binding DN does not have authority to do the operation (such as an add or delete) that the client requests. To get information about the error, look at the QDIRSRV job log.

### ***[iSeries server]: A remote host refused an attempted connect operation***

The most common causes of this error include:

- ▶ An LDAP client makes a request before the LDAP server on the specified iSeries server is up and in select wait status.
- ▶ The user specifies a port number that is not valid. For example, the LDAP server is listening on port 386, but the client request attempts to use port 387.

To get information about the error, look at the QDIRSRV job log.

### ***[iSeries server]: Failed to connect to SSL server***

This error occurs when the LDAP server rejects the client connection because a secure socket connection cannot be established. This can occur when the Certificate Management support rejects the client's attempt to connect to the server. Use Digital Certificate Manager to make sure that your certificates are set up properly, then try to connect again.

## 4.3 Configuring WebSphere and Domino to use OS/400 LDAP

This section describes configuring WebSphere and Domino to use the OS/400 LDAP server for user authentication. To complete the examples in this section you should complete the configuration of the OS/400 LDAP server as described in Section 4.2, “Using OS/400 LDAP” on page 60.

Similar to Domino, WebSphere Application Server does not by default require users to enter a user name and password to access any of the resources it manages. Several steps need to be performed to activate security. With WebSphere Application Server, you can secure your applications by configuring the following security policies:

- ▶ Authentication: determination of who is making a request
- ▶ Authorization: determination of whether a request is to be honored
- ▶ Delegation: determination of who will be handling the request

To restrict the use of the WebSphere Application Server by allowing only *authenticated users* access to its resources, a user registry (a directory of valid users) needs to be in place. This can either be the user registry provided by the platform on which WebSphere is running (for OS/400 this would be the user profiles), or any LDAP server. On an iSeries server, an LDAP server can be either SecureWay Directory for OS/400 or Lotus Domino for iSeries.

If you plan to use the same user registry for multiple servers and later enable single sign-on, you *must* use an LDAP server for authentication.

In this section you will learn how to enable security for your WebSphere Application Server using OS/400 LDAP in order to authenticate users before giving them access to resources and also how to configure Domino to use OS/400 LDAP.

### 4.3.1 Authentication concepts of the WebSphere Application Server

Authentication policy determines how authentication is accomplished. Authentication is performed in two steps:

- ▶ Acquiring the authentication data of a principal.
- ▶ Validating the authentication data against a user registry.

An authentication mechanism validates authentication data against an associated user registry. WebSphere Application Server offers these choices for authentication mechanisms:

- ▶ Lightweight Third Party Authentication (LTPA)
- ▶ Native OS

#### ***Lightweight Third Party Authentication (LTPA)***

LTPA uses a trusted third-party server to authenticate the user. This is the heart of the distributed security model. Since all application servers trust the third-party authentication server, security information can be passed along with requests from one application server to another. Use the LTPA authentication mechanism when applications are distributed across multiple application servers.

#### ***Native OS***

The native OS authentication mechanism uses native OS/400 routines (OS/400 user profiles) to authenticate the user. Native OS is easier to configure than LTPA, but can be used for only the simplest topologies. Note that authenticating through the native OS mechanism *does not* log the user onto the iSeries server. Even though user profiles and passwords are used for authentication, no jobs or threads are executed under the users' profiles.

A challenge type specifies how a server will challenge and retrieve authentication data from a user. The choices for challenge type are:

- ▶ **None:** The user is not challenged for authentication data. If the requested resource is protected, then the user will not be served the resource.
- ▶ **Basic:** The user is challenged for a user ID and password.
- ▶ **Custom:** Applicable only to Web clients. The custom challenge type is used when one wants to configure the server to use a customized HTML form to retrieve the user ID and password.
- ▶ **Certificate** (new with v3.5): Applicable only to Web clients. The user is required to present a digital certificate (X.509) to establish the connection. With the certificate challenge type, the Web server is trusted to authenticate the user through the SSL exchange. Then for authorization purposes, the WebSphere security infrastructure identifies the principal by extracting information from the certificate and mapping it to an entry in the user registry.

A user registry is where the user and group information is stored. It contains a mapping of principals to authentication information and privilege attributes such as access ID, password and group IDs. A “principal” is a representation of a human user or system entity such as a server process. The choices for user registry are:

- ▶ Native OS - OS/400 profiles
- ▶ Lightweight Directory Access Protocol (LDAP)

### 4.3.2 Enabling global security in the WebSphere Application Server

To configure the WebSphere Application Server to use the OS/400 LDAP server, you must enable global security. This is done via the WebSphere Administrative Console. To enable global security and configure WebSphere to use the OS/400 LDAP server, perform the following steps:

1. From the WebSphere Administrative Console, click the **Wizard** icon and select **Configure Global Security Settings** (see Figure 4-14).

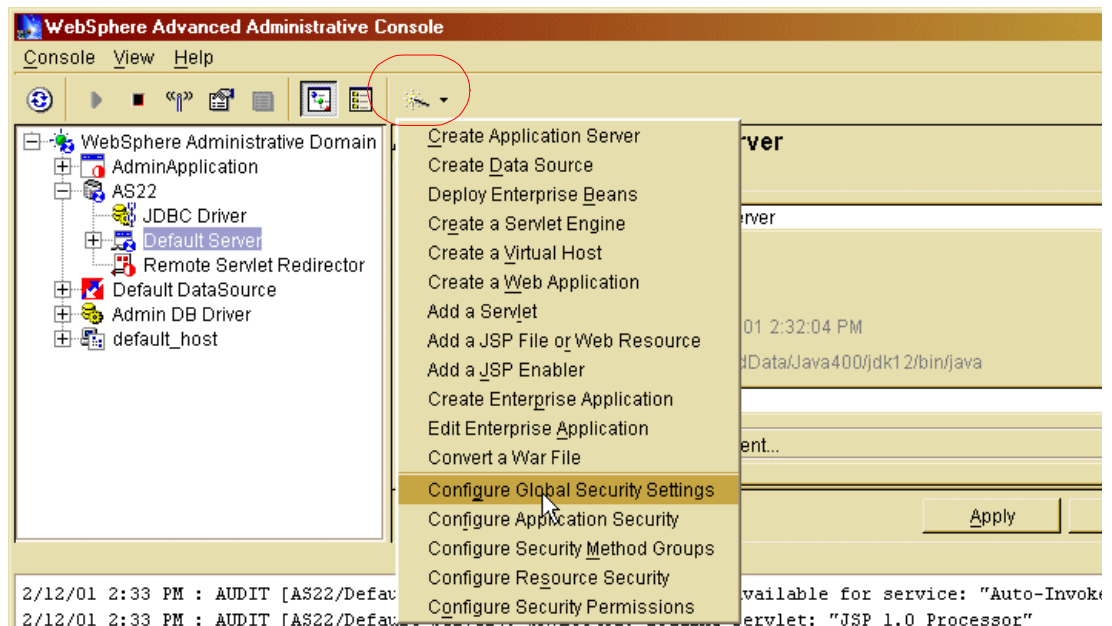


Figure 4-14 Configure Global Security Settings for WebSphere

2. The Set Global Security wizard starts. On the first pane of the General tab, select **Enable Security**. Verify that the Security Cache Timeout is set to a reasonable value. When the timeout is reached, WebSphere Application Server clears the security cache and rebuilds the security data. If the value is set too low, the extra processing overhead may be unacceptable. If the value is set too high, you create a security risk by caching security data for a long period of time. The default value is 600 seconds. See Figure 4-15. Click **Next**.

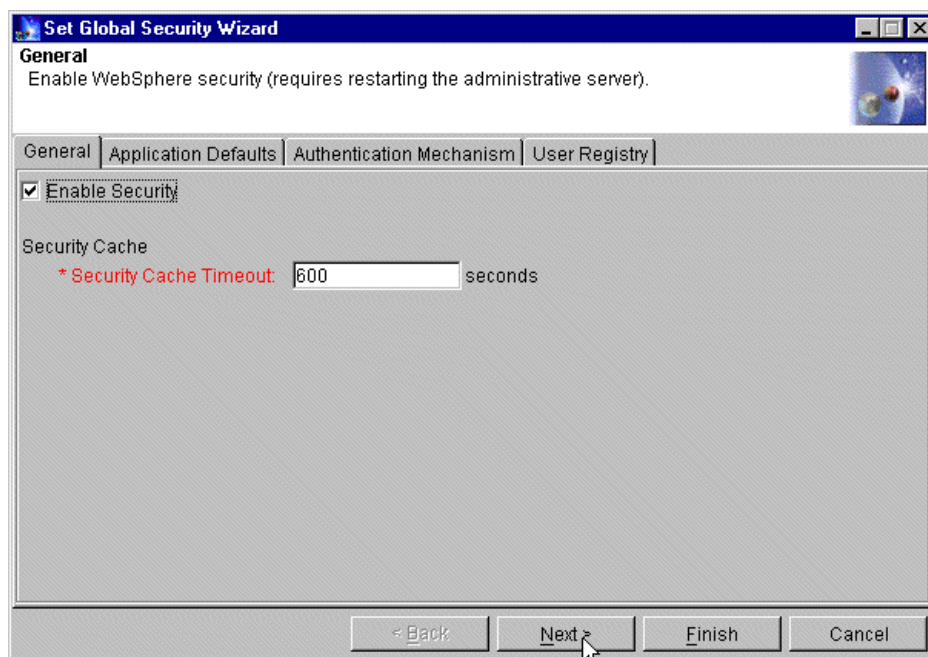


Figure 4-15 Set Global Security Wizard - General tab

3. The Application Defaults tab is displayed. Set the Realm Name field to your domain portion of your fully qualified Internet name for the system running your WebSphere administrative domain. In our example the realm is itsoroch.ibm.com. This means security will apply to machines in this realm such as as22.itsoroch.ibm.com.

For the Challenge Type, select **Basic (User ID and Password)**, as shown in Figure 4-16. Click **Next**.

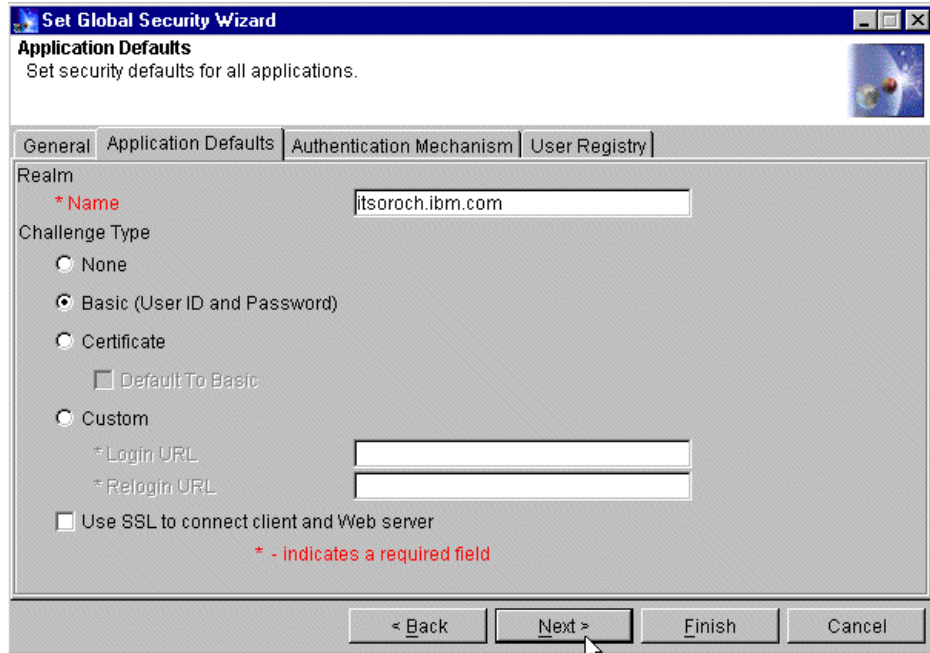


Figure 4-16 Set Global Security Wizard - Application Defaults tab

4. The Authentication Mechanism tab is displayed. Select **Lightweight Third Party Authentication (LTPA)**, as shown in Figure 4-17. Click **Next**.

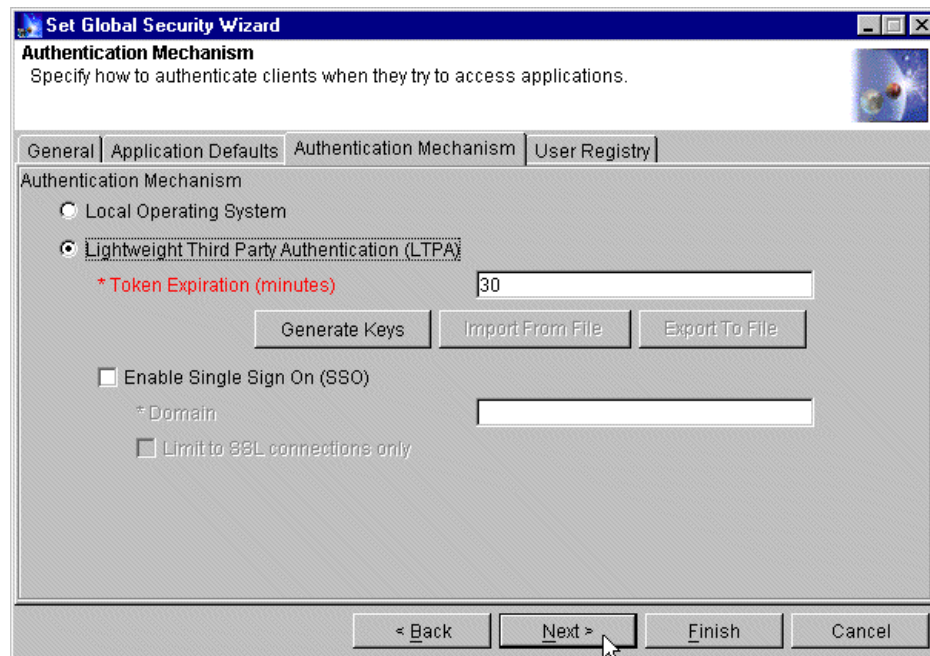


Figure 4-17 Set Global Security Wizard - Authentication Mechanism tab

5. The User Registry tab is displayed. This is where you tell WebSphere that you want to use OS/400 LDAP as the user registry. WebSphere will then use LDAP to connect to the OS/400 LDAP server and authenticate users. The OS/400 LDAP server must be running, since the wizard will attempt to connect to the LDAP server to verify the information you enter.

Fill in the fields as explained in the following list:

- **Security Server ID**

This is the user ID of the Administrator for the WebSphere administrative domain. This user ID is used later when accessing WebSphere administration services using the WebSphere Administrative Console. By default, this should be the value of the short name or user ID for a user already defined in the LDAP directory. Do not specify a Distinguished Name by using `cn=` or `uid=` before the value.

**Note:** Later, when you restart the WebSphere Administrative Console, you must enter the value exactly as you specified it in this field.

- **Security Server Password**

This is the valid password for the Security Server ID. This field is case sensitive.

- **Directory Type = SecureWay**

This value should be set for the type of LDAP server you are using. For example, select **SecureWay** for IBM SecureWay LDAP directory, or **Domino 5.0** for the Domino LDAP directory.

- **Host**

This is the fully qualified host name on which the LDAP directory is running.

- **Port**

This is the port on which the LDAP directory runs. You may leave this field blank for the default, non-SSL (Secure Sockets Layer) port of the LDAP directory (port 389).

- **Base Distinguished Name**

This is the Distinguished Name (DN) of the directory in which searches begin within the LDAP directory. For example, for a user with a DN of `cn=John Doe, ou=Rochester, o=IBM, and c=US`, you could specify a base DN of `ou=Rochester, o=IBM, c=US` or `o=IBM, c=us` or `c=us`. This is a required field for all LDAP directories except the Domino Directory.

**Note:** If you are using the Domino Directory, and you specify a Base Distinguished Name, you cannot grant permissions to individual Web users for resources managed by your WebSphere application server.

- **Bind Distinguished Name**

This is the DN of the user who is capable of performing searches on the directory. In most cases, this field is not required, since all users are usually authorized to search an LDAP directory. However, if the LDAP directory contents are protected from all LDAP users, you need to specify the DN of an authorized user, such as the administrator of the directory (for example, `cn=Administrator`).

- **Bind Password**

This is the valid password for the user specified as the Bind Distinguished Name. This is required only if you specify a value for Bind Distinguished Name. This field is case sensitive.

Complete the fields as shown in Figure 4-18. At a minimum, you will need to change the Security Server ID, Security Server Password, Directory Type, and Host fields to match your environment.

Click **Finish** to save the Global Security Settings.



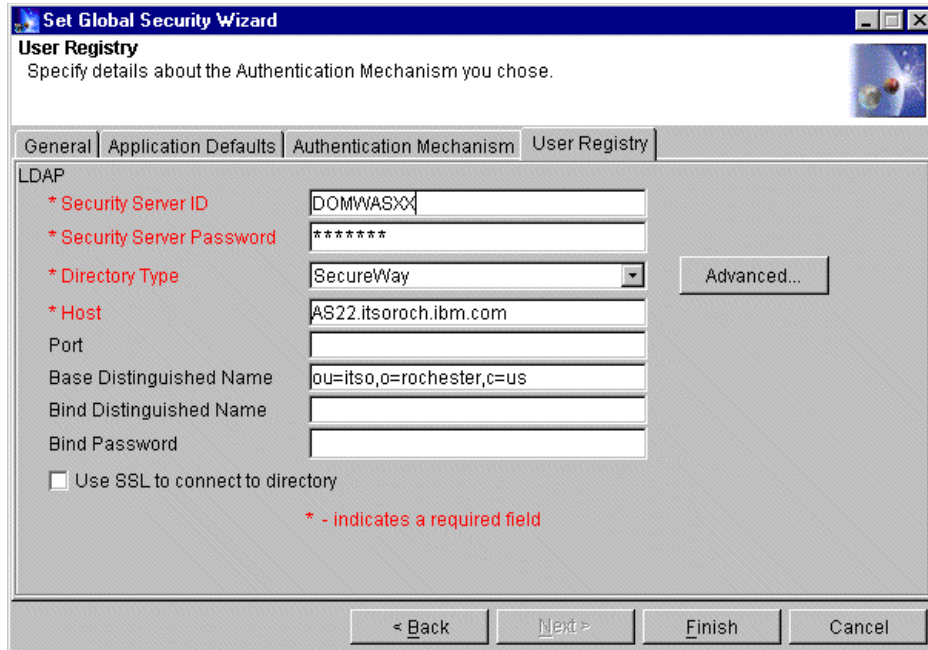


Figure 4-18 Set Global Security Wizard: User Registry tab

6. You are then prompted for an LTPA password as shown in Figure 4-19. This is the password for the keys that WebSphere is generating. Enter a password and click **OK**.

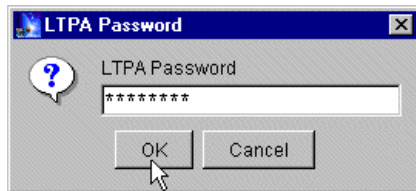


Figure 4-19 LTPA Password

7. After a couple of seconds, a message box appears (see Figure 4-20) stating that your changes will not take effect until the WebSphere Administrative server is restarted. Click **OK** to continue.

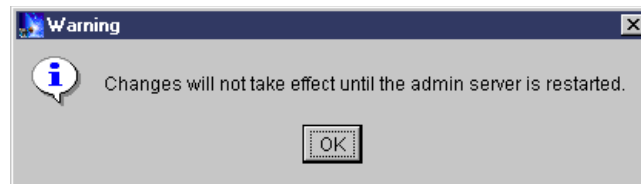


Figure 4-20 Global Security Settings finished successfully message

8. To restart the WebSphere Administrative server, right-click the node and select **Restart** (see Figure 4-21).

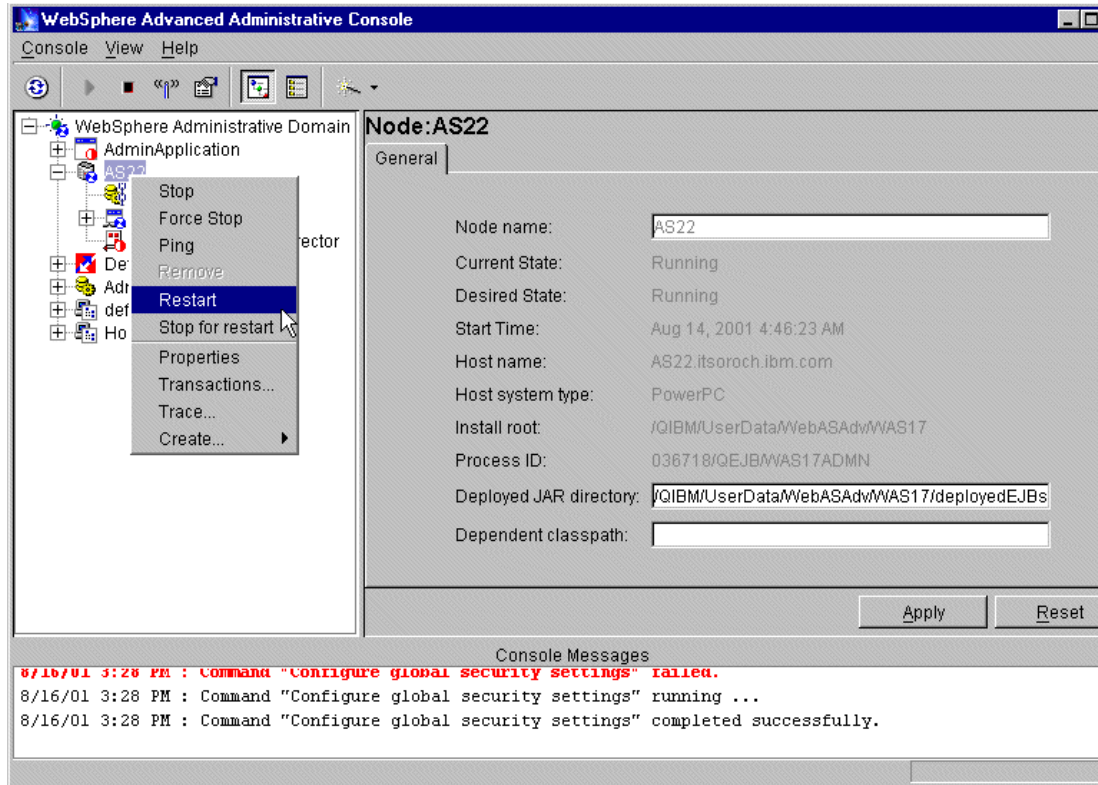


Figure 4-21 Restarting the WebSphere Administration server

9. A warning message appears (see Figure 4-22) stating that you are trying to stop the node to which the console is connected. Click **Yes** to continue.

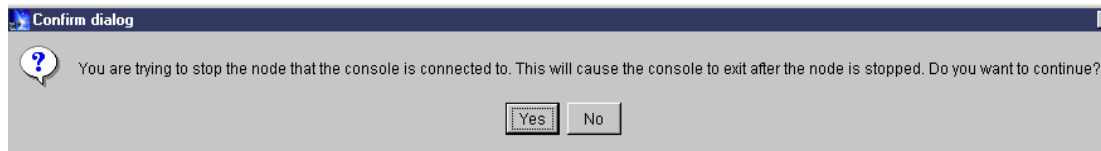


Figure 4-22 Restart WebSphere warning message

10. Make sure that your WebSphere monitor job (WASxxMNTR) and the administrative server job (WASxxADMN), where WASxx is the name of your WebSphere Application Server instance, is completely restarted before continuing. To check this, perform the following tasks:
  - a. Enter the Work with Active Jobs (WRKACTJOB) command to verify that these jobs are in JVAW and EVTW status:
 

```
wrkactjob sbs(qejbsbs)
```

Monitor the administrative server task (or job) (WASxxADMN) from the Work with Active Jobs window (WRKACTJOB) to ensure that the WebSphere Administrative server restarts successfully. As you watch the Administrative server job, notice that it stops, starts, stops, and then starts again. This is expected after Global Security Settings have been enabled or changed.
  - b. Display the job log for the administrative server job (WASxxADMN). From the WRKACTJOB display, enter option 5 (Work with) next to the WASxxADMN job. Press Enter. Note, it may take several minutes - possibly more than 10 minutes - depending on your system.

- c. On the Work with Job display, enter option 10 (Display job log, if active or on job queue) to display the job log. Press Enter.
- d. Look for the following message:  
WebSphere administration server WASxxADMN ready.  
You may have to press F5 several times to refresh the display and scroll down if you see More... in the lower right corner of your display.
- e. Position the cursor on the ready message (step d). Press F1 to verify that the administration server is listening to the correct port assigned to your team (9xx) and that the message is new (look at the time stamp). See Figure 4-23.

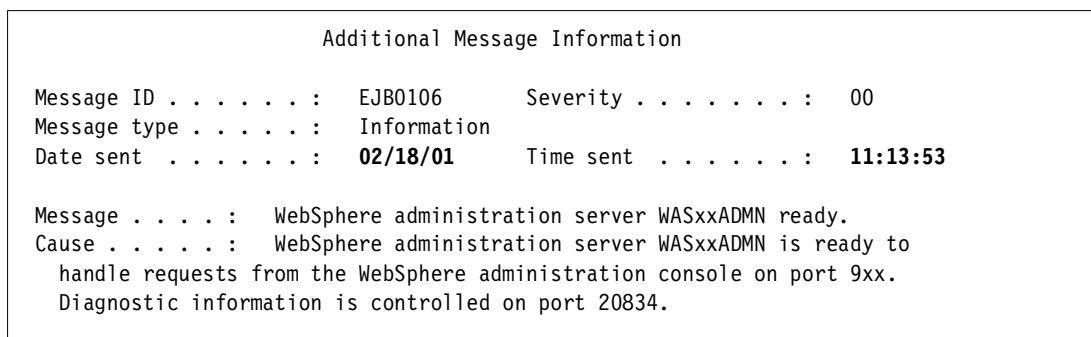


Figure 4-23 WebSphere administration server started successfully message

11. Once the WebSphere Administrative Server has been successfully and completely restarted, restart the WebSphere administrative console and connect it to your WebSphere instance.
12. Notice that you now need to sign on to the WebSphere Administrative console because you have enabled security. Specify the user ID and password exactly as you configured previously for the Security Server ID and Security Server Password fields in the Global Security Settings wizard (see Figure 4-24).

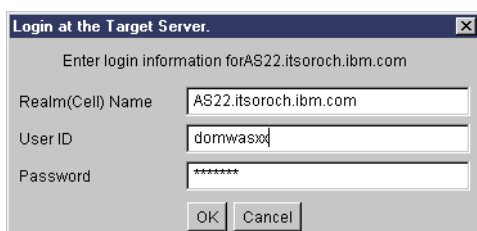


Figure 4-24 Sign on to the WebSphere Administration Console

13. Wait until the WebSphere Administrative message console displays the Console Ready message before continuing.

This completes the configuration of WebSphere to use the OS/400 LDAP server for authentication. In order to test this, you need to apply WebSphere security to some WebSphere components. See Section 4.6, “Securing WebSphere resources” on page 109 for an example of how you can secure WebSphere resources.

### 4.3.3 Creating a Directory Assistance Domino database

For Domino to use the OS/400 LDAP server, instead of its own LDAP, you must create a Directory Assistance database that points to the OS/400 LDAP server. To create a Directory Assistance database in Domino, perform the following steps:

1. From your Lotus Notes client pull-down menu, select **File -> Database -> New**.
2. From the New Database window, enter information in the Server, Title, and File Name fields. In our example, we entered the following values (see Figure 4-25):
  - Server = DomWASxx/Domxx
  - Title = Directory Assistance
  - File Name = Director.nsf
3. Click the **Template Server** button, and select your Domino server. For our example the Domino server was DomWASxx/Domxx.
4. Select the **Directory Assistance** template (DA50.NTF).
5. Make sure **Inherit future design changes** check box is selected, and click **OK**.

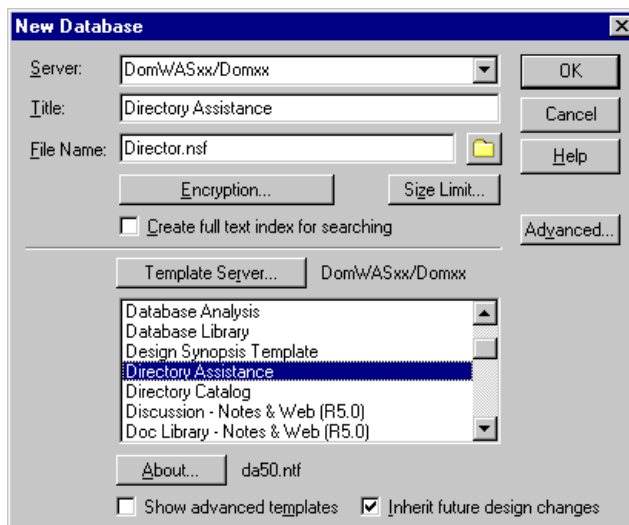


Figure 4-25 Creating a Directory Assistance database

6. When the *About Directory Assistance* document appears, press Esc.
7. You are now in the Directory Assistance database. Click the **Add Directory Assistance** button (see Figure 4-26).

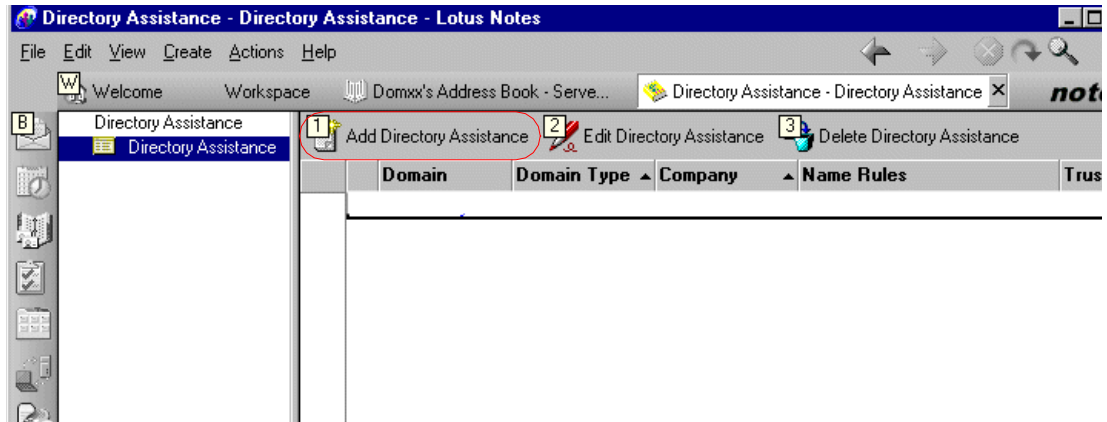


Figure 4-26 Domino Directory Assistance database

8. In the Directory Assistance document, click the **Basics** tab and fill in the fields with the following values (see Figure 4-27):

- Domain type: **LDAP**
- Domain name: **SecureWay**
- Company name: **IBM**
- Search order: leave this field blank
- Group expansion: **Yes**
- Nested group expansion: **Yes**
- Enabled: **Yes**

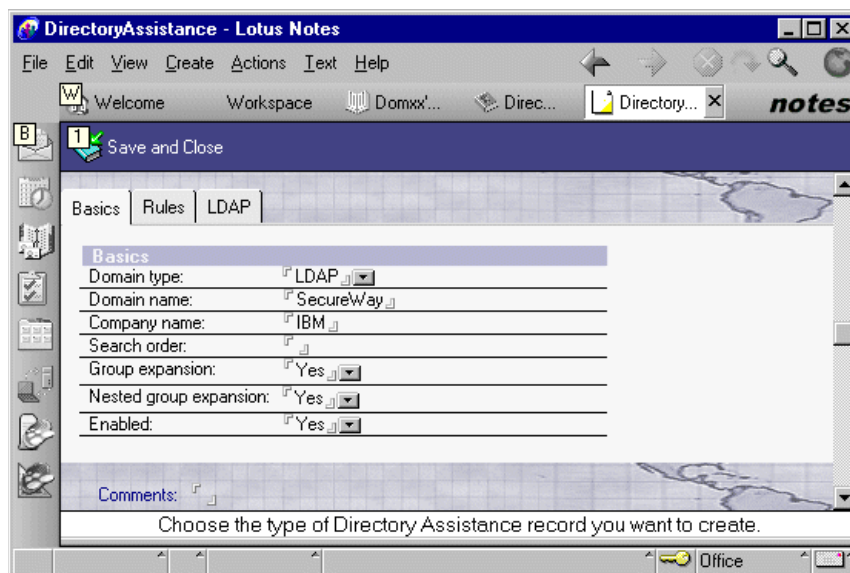


Figure 4-27 Directory Assistance: Basics tab

9. Click the **Rules** tab and fill in the fields with the following values (see Figure 4-28):

- For Rule 1, ensure that Enable is set to **Yes**.
- For Rule 1, ensure that Trusted for Credentials is set to **Yes**.

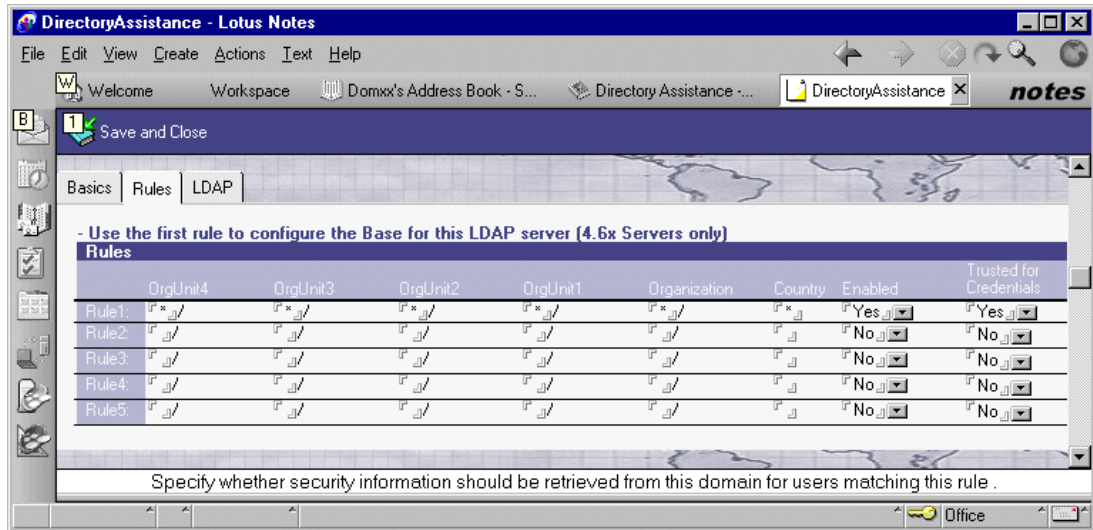


Figure 4-28 Directory Assistance: Rules tab

10. Click the **LDAP** tab and fill in the fields with the following values (see Figure 4-29):

- Hostname: Your fully qualified iSeries server host name. For our example this is AS22.itsoroch.ibm.com.
- Optional Authentication Credentials Username: Your OS/400 LDAP administrator. For our example this is cn=Administrator.
- Optional Authentication. Credentials Password: Your OS/400 LDAP administrator's password. For our example, this is ldappw.
- Base DN for search: Your OS/400 LDAP schema. For our example, this is ou=itso,o=rochester,c=us.
- Perform LDAP search for: Choose **NotesClients/WebAuthentication**.
- Channel encryption: Choose **None**.
- Leave all other fields as default.

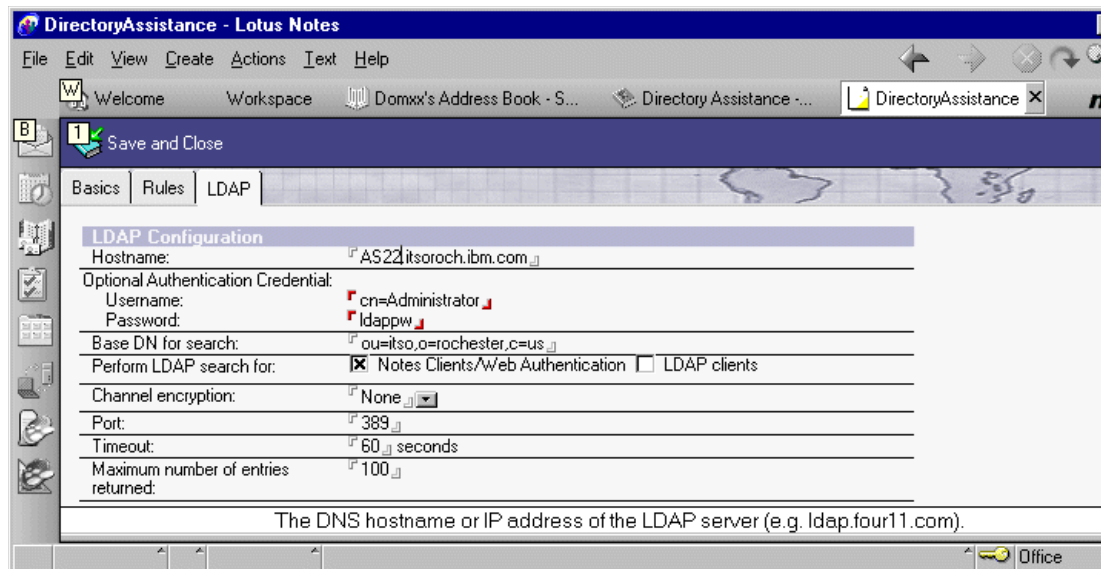


Figure 4-29 Directory Assistance: LDAP tab

11. Click **Save and Close** to exit and save this document.

#### 4.3.4 Configuring Domino to use OS/400 LDAP

To configure the Domino server to use OS/400 LDAP, you must update the Domino server document. Perform the following steps:

1. From your Lotus Notes client, open the Domino Directory, and click the **Server** twistie. Select the **Servers** view (see Figure 4-30).

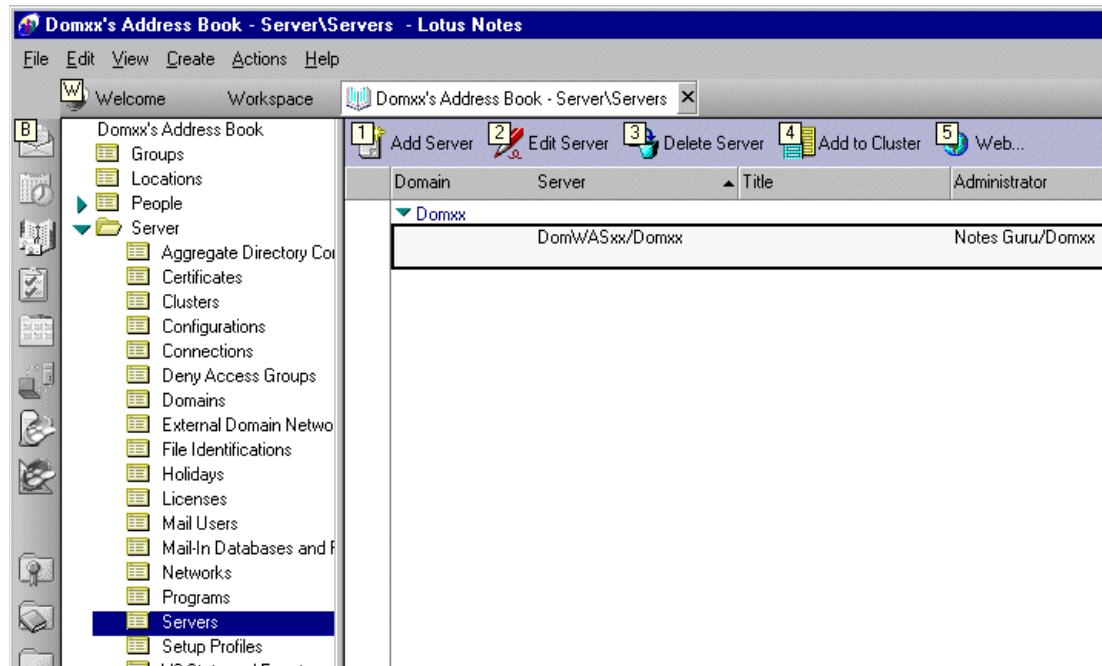


Figure 4-30 Domino server view

2. Select your Domino server, and click the **Edit Server** button.
3. In the Domino server document, click the **Basics** tab. In the Directory Assistance database name field, type the filename of your Directory Assistance database that you created in Section 4.3.3, "Creating a Directory Assistance Domino database" on page 88. For our example this was `Director.nsf` (see Figure 4-31).



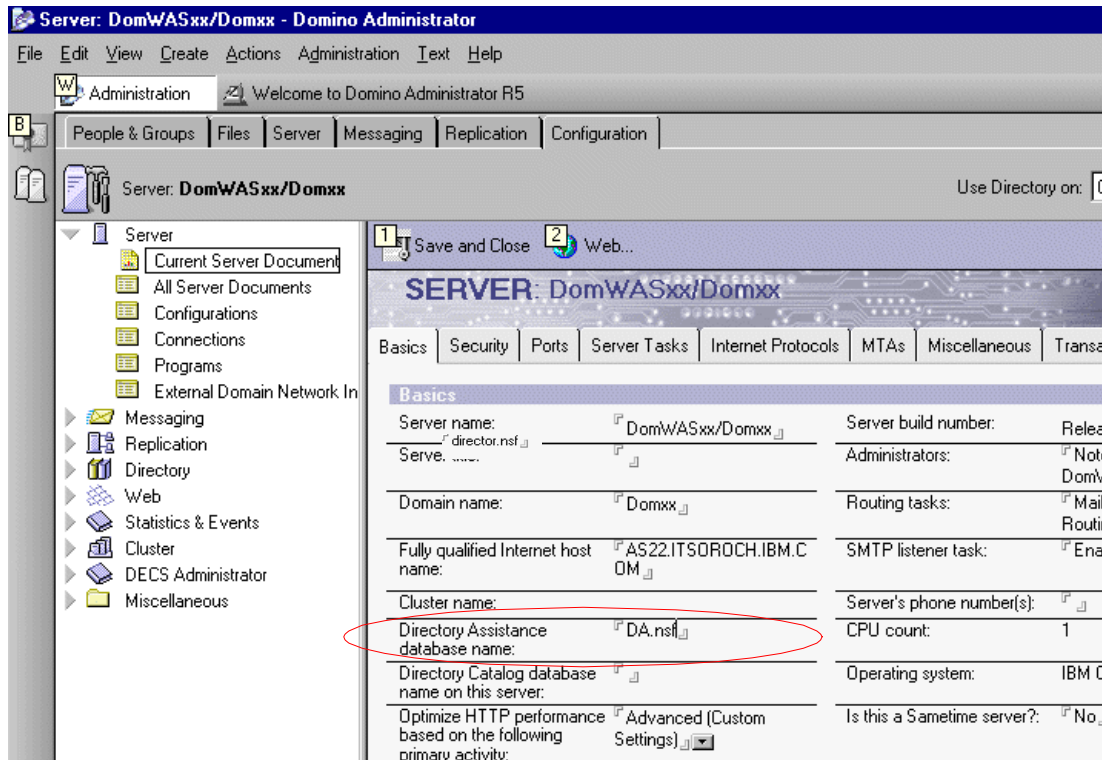


Figure 4-31 Adding the Directory Assistance database name to the Domino server

4. Select the **Ports** tab, and then select the **Internet Ports** sub-tab. Then, select the **Directory** sub-sub-tab (see Figure 4-32).



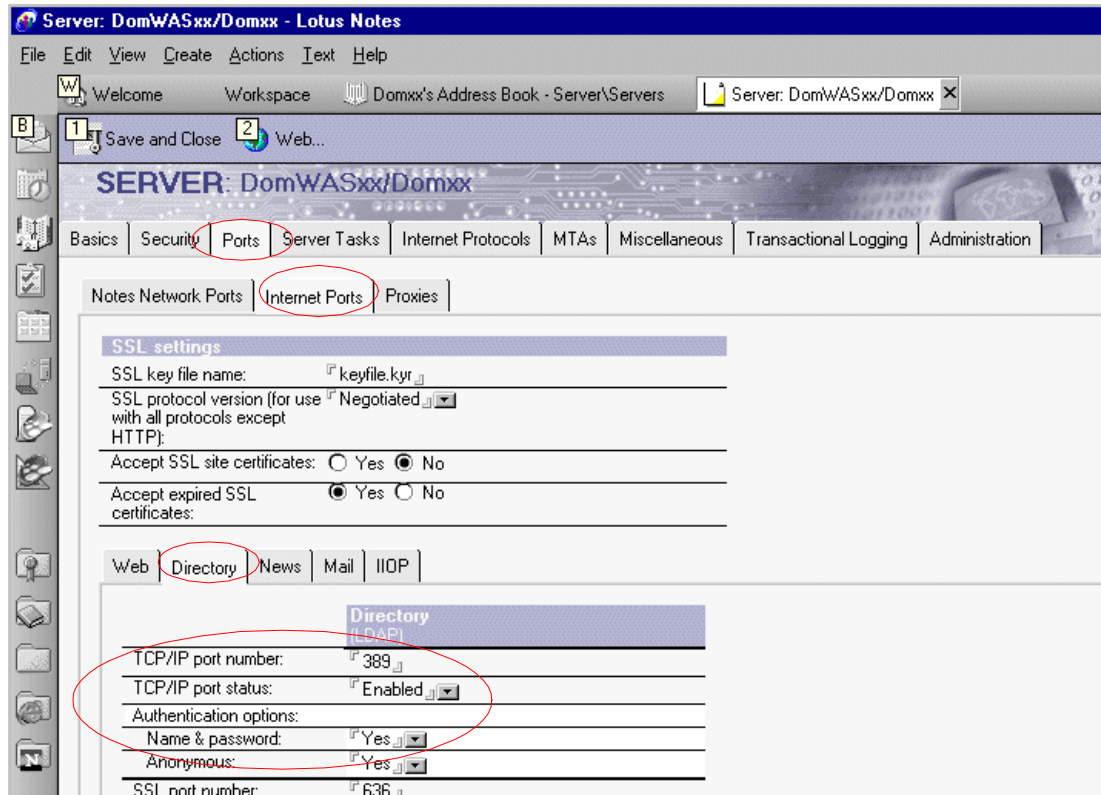


Figure 4-32 Changing the directory port

5. Change the TCP/IP port number to the OS/400 LDAP port and ensure that the Name & Password field in the TCP/IP Authentication options is set to **Yes**.
6. Click the **Save and Close** button to save and exit the Domino server document.

## Updating Domino database ACLs

You now may need to add your OS/400 user profiles that have been published to your OS/400 LDAP directory to your Domino database ACLs. Perform the following steps:

1. From your Lotus Notes client workspace, right-click the Domino database icon whose ACL you would like to change and select **Database -> Access Control**.
2. From the Access Control List window, click the **Add** button to add a new user to the database ACL. On the Add User window, enter the information for your OS/400 LDAP directory user in the following format:

cn=DOMWASxx/ou=ITS0/o=IBM/c=US

Click **OK** (see Figure 4-33).

**Note:** The syntax here is different from the LDAP syntax because a forward slash (/) is used instead of the comma (,).

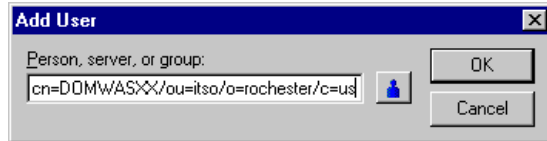


Figure 4-33 Adding a user in LDAP format

- When a user is added, the format appears different from what you entered, that is the hierarchy identifiers (o=, ou=, c=) disappeared. This is expected. Make sure the new ACL member has the correct authority for the database. Click **OK** to exit the Access Control List window (see Figure 4-34).

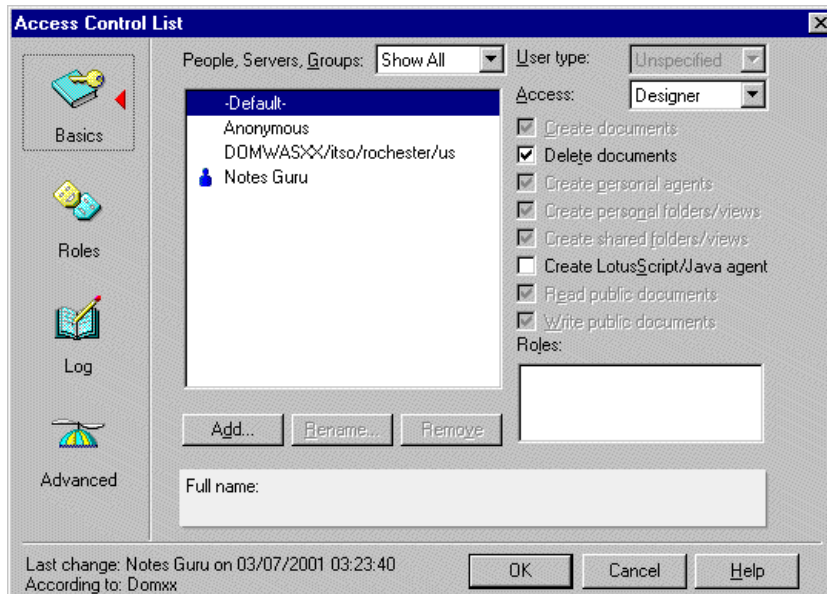


Figure 4-34 New OS/400 user added to ACL

- For these changes to take effect, the Domino server must be stopped and restarted. To do this, from your 5250 session, enter the following OS/400 CL command and press Enter:  
WRKDOMSVR DomWASxx
- From the Work with Domino Servers window, enter option 8 (Work Console) next to your Domino server.
- On the Work with Domino Console window, enter the following command:  
restart server
- Press F5 until you see the Domino server restarted successfully.

## 4.4 Using Domino LDAP

One aspect of the Domino and WebSphere security models, the directory (referred to as the “user registry” in WebSphere), can be made common. This is possible since both Domino and WebSphere support the Lightweight Directory Access Protocol (LDAP) for directory access. The Domino Directory component provides for LDAP access and WebSphere can be set up to use an LDAP accessible directory as its user registry.

In this section we discuss configuring the Domino Directory as the user registry for Web user authentication and authorization in a combined WebSphere and Domino environment on the iSeries server. This involves having Domino set up to act as an LDAP server, and ensuring that a user in the Domino Directory is set up as Administrator and has an Internet password. Section 4.5, “Configuring WebSphere to use Domino LDAP” on page 101 then covers how to setup WebSphere to use Domino LDAP as the common user registry.

#### 4.4.1 Configuring the Domino Directory for LDAP access

Domino R5 supports access to the Domino Directory via LDAP. If you configured the Domino server with the Directory Services parameter set to \*LDAP, no further action is required, since the configuration automatically changes the Domino Server configuration document and adds LDAP to the Server Tasks line in the Domino server’s notes.ini file.

It is also possible to extend the Domino LDAP schema if necessary. We did not need to do this to support our testing for this redbook. If you need to do so, consult the *Domino R5 Administration Help* and *Administrator’s Guide* for further direction.

You also have to make sure that there is a user defined in Domino Directory that has an Internet password. In our case we created a Domino administrator ID called “WASAdmin” and assigned an Internet password to it.

If you did not originally configure the Domino server for LDAP, then you can either use the OS/400 CL command of Change Domino Server (CHGDOMSVR) to change the Domino server configuration, or you can manually change the Domino server configuration document using the following steps:

1. Start the Domino R5 Administrator client.
2. Open your Domino server and select the **Configuration** tab.
3. Select **Server -> Current Server Document** to display the Domino server document. Click the **Edit Server** button.
4. Click the **Ports** tab.
5. Click the **Internet Ports** and **Directory** sub-tabs to display the LDAP port settings as shown in Figure 4-35.

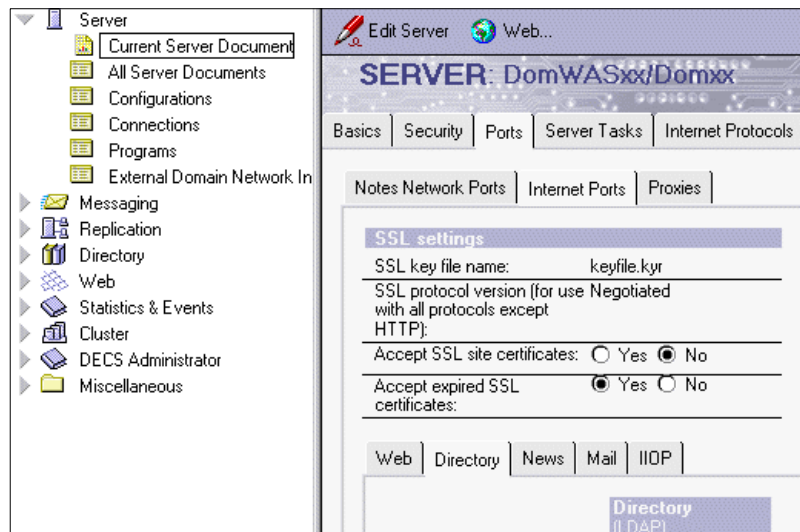


Figure 4-35 Domino Administrator client showing tabs to Directory settings

6. The default LDAP standard and SSL port numbers (389 and 636) are shown. Ensure the TCP/IP port status is set to **Enabled** and the port number to **389** or another port number based on your environment. By default both the Domino and OS/400 LDAP servers listen to port 389. See Figure 4-36.

Click **Save and Close** to save your changes.

**Recommendation:** For a production environment, we recommend that the TCP/IP port be disabled and the SSL port be enabled with at least name & password authentication.

The screenshot shows the Domino Directory configuration window. The top bar includes 'Edit Server' and 'Web...' buttons. The title bar reads 'SERVER: DomWASxx/Domxx'. Below the title bar are tabs for 'Basics', 'Security', 'Ports', 'Server Tasks', 'Internet Protocols', 'MTAs', 'Miscellaneous', and 'Transactional Logging'. The 'Ports' tab is selected, showing sub-tabs for 'Notes Network Ports', 'Internet Ports', and 'Proxies'. The 'Internet Ports' sub-tab is active, displaying 'SSL settings' and 'Directory (LDAP)' sections.

**SSL settings**

SSL key file name:	keyfile.kyr
SSL protocol version (for use Negotiated with all protocols except HTTP):	
Accept SSL site certificates:	<input type="radio"/> Yes <input checked="" type="radio"/> No
Accept expired SSL certificates:	<input checked="" type="radio"/> Yes <input type="radio"/> No

**Directory (LDAP)**

TCP/IP port number:	389
TCP/IP port status:	Enabled
Authentication options:	
Name & password:	Yes
Anonymous:	Yes
SSL port number:	636
SSL port status:	Disabled
Authentication options:	
Client certificate:	No
Name & password:	No
Anonymous:	Yes

Figure 4-36 LDAP port settings in the Domino Directory

7. You should also check to ensure the Administrator has an Internet password, as seen in Figure 4-37.

**PERSON: Notes Guru/Domxx** Notes Guru/Domxx @ Domxx

Basics | Mail | Work/Home | Other | Miscellaneous | Certificates | Administration

**Name**

First name: [Notes]

Middle initial: [ ]

Last name: [Guru]

User name: [Notes Guru/Domxx  
Notes Guru  
Postmaster]

Alternate name: [ ]

Short name/UserID: [nguru]

Personal title: [ ]

Generational qualifier: [ ]

Internet password: [355E98E7C7B59BD810ED845AD0FD2FC4]

Figure 4-37 Administrator's Internet password

This completes the setup of the Domino LDAP service.

## 4.4.2 Starting and stopping the Domino LDAP task

The Domino LDAP task can be started in a number of different ways:

- If the LDAP server task was not already running, you can start it from the Domino Administrator client. To start the LDAP task from the Domino Administrator client, perform the following steps:
  - a. From the Domino Administrator client, click the **Server** tab and then the **Status** sub-tab.
  - b. Click the **Task** button on the right side of the window and click **Start** (see Figure 4-38).

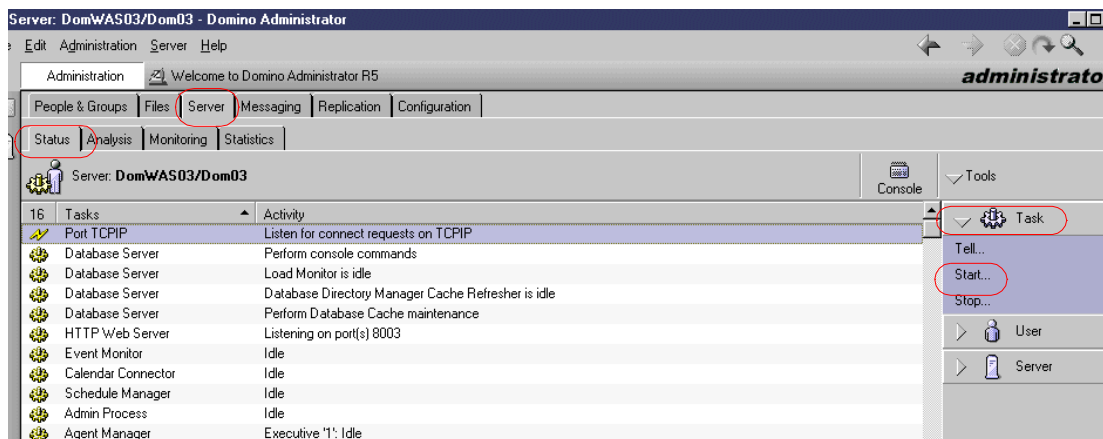


Figure 4-38 Domino server Status view

- c. From the Start New Task window, scroll down and select the **LDAP Server** task, and click the **Start Task** button to start the LDAP server. See Figure 4-39.

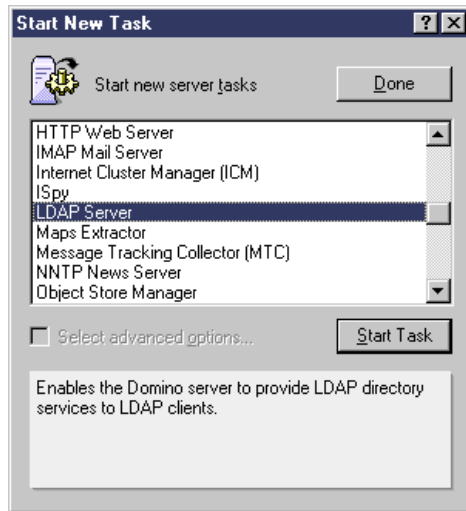


Figure 4-39 Start LDAP server task

- d. Click **Done** to close the dialog window.
- If the LDAP server task is already running, you should restart it after making the changes to the Domino server document. To restart the LDAP task from the Domino Administrator client, perform the following steps:
  - a. Select the LDAP task on your Domino server from the **Server** tab and then the **Status** sub-tab.
  - b. Right-click the highlighted LDAP server task and click **Tell Task**. See Figure 4-40.

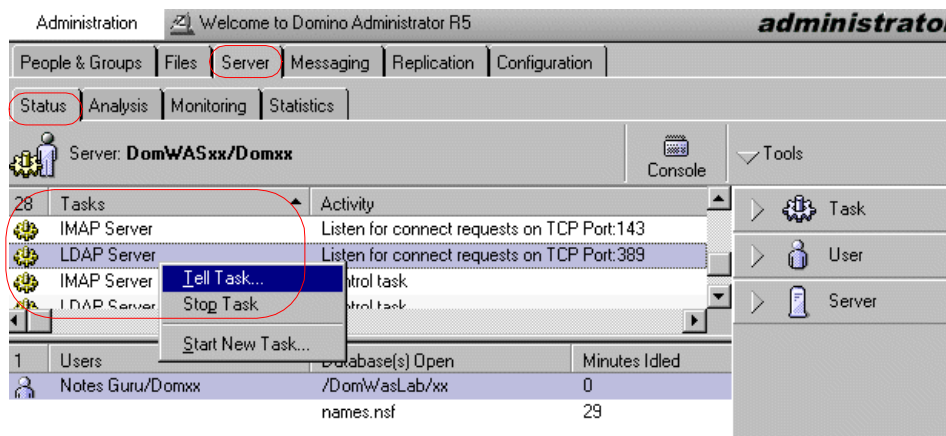


Figure 4-40 Selecting the LDAP server task

- c. On the Tell LDAP server window, select the **Reload schema** check box and then click **OK**. See Figure 4-41.

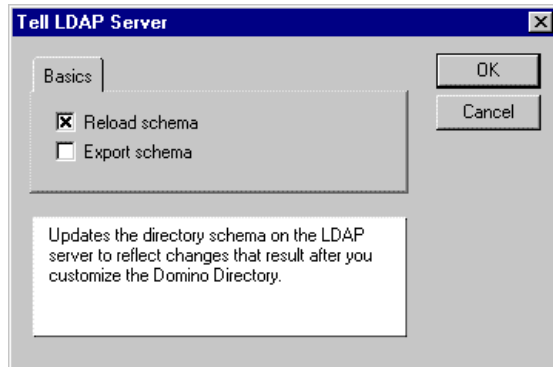


Figure 4-41 Reload schema

- ▶ The LDAP server task can be started from the Domino server console by entering the **load ldap** command.
- ▶ You can add the LDAP server task name to the Server Tasks= variable in the Domino server's notes.ini file to start the LDAP task when the Domino server is started. If you selected LDAP support during the configuration of the Domino server this entry is already in the notes.ini file.

The Domino LDAP server task can be stopped in several different ways:

- ▶ To end the LDAP task from the Domino Administrator client, perform the following steps:
  - a. From the Domino Administrator client, click the **Server** tab and then the **Status** sub-tab.
  - b. Click the **Task** button on the right side of window and click **Stop** (see Figure 4-42).

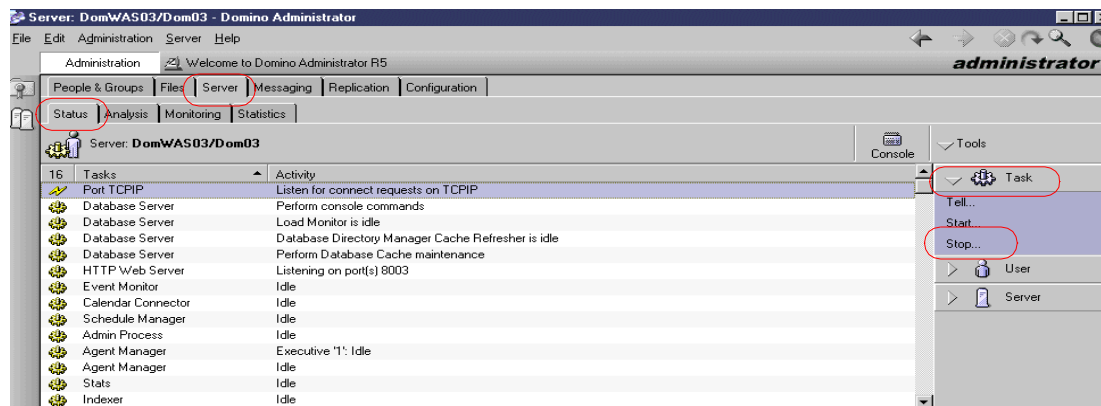


Figure 4-42 Domino server Status view

- c. From the Stop Task window, scroll down and select the **LDAP Server** task, and click the **Stop Task** button to end the LDAP server.
- ▶ The LDAP server task can be stopped from the Domino server console by entering the **tell ldap quit** command.

#### 4.4.3 Verifying the connection to the Domino LDAP server

Once you have configured and started the LDAP service on your Domino server, you should verify the LDAP task is active and check the connection to the Domino LDAP server.

Perform the following steps to verify the Domino LDAP server task is running:

1. From a 5250 emulation session, enter the following command on the OS/400 command line and press Enter:  
WRKDOMSVR DomWASxx
2. Enter option 8 (Work with console) in the Opt field next to your Domino server. Press Enter.
3. On the command line at the bottom of the Work with Domino Console window, enter either the **sh ta** command, or its long form, **show tasks**, and press Enter.
4. Scroll down until a window similar to the one shown in Figure 4-43 appears.

```
Work with Domino Console
Server: DomWASxx

Previous subcommands and messages:
Schedule Manager      Idle
Admin Process         Idle
Agent Manager         Executive '1': Idle
Agent Manager         Idle
Stats                Idle
Indexer              Idle
Router              Idle
Replicator           Idle
Billing              Idle
HTTP Web Server       Listening on port(s) 80xx
LDAP Server          Listen for connect requests on TCP Port:389xx
LDAP Server          Control task
DECS Server          Idle

Enter a Domino subcommand.
====> _____

F3=Exit  F5=Refresh  F6=Print  F9=Retrieve
F17=Top  F18=Bottom F21=Command line
```

Figure 4-43 Work with Domino Console: show tasks command

5. Make sure the LDAP server is actually listening to the LDAP port that you assigned. Press F3 to exit the Domino console.

To check the connection to the Domino LDAP server, there are several methods for doing this. Here we explain how to use a Web browser and the QShell utility.

### Using a Web browser

To verify the connection to the Domino LDAP server, you need an LDAP client. Netscape Navigator is able to look up entries in a directory using LDAP. Perform the following steps to test the Domino LDAP server:

1. Start Netscape Navigator and enter the following URL. You will need to change to the Domino server name from the examples of domwasxx and cn=Notes Guru, o=Domxx to match your environment.  
ldap://domwasxx/CN=Notes Guru, O=Domxx
2. Netscape looks up the directory entry and displays the information as shown in Figure 4-44.



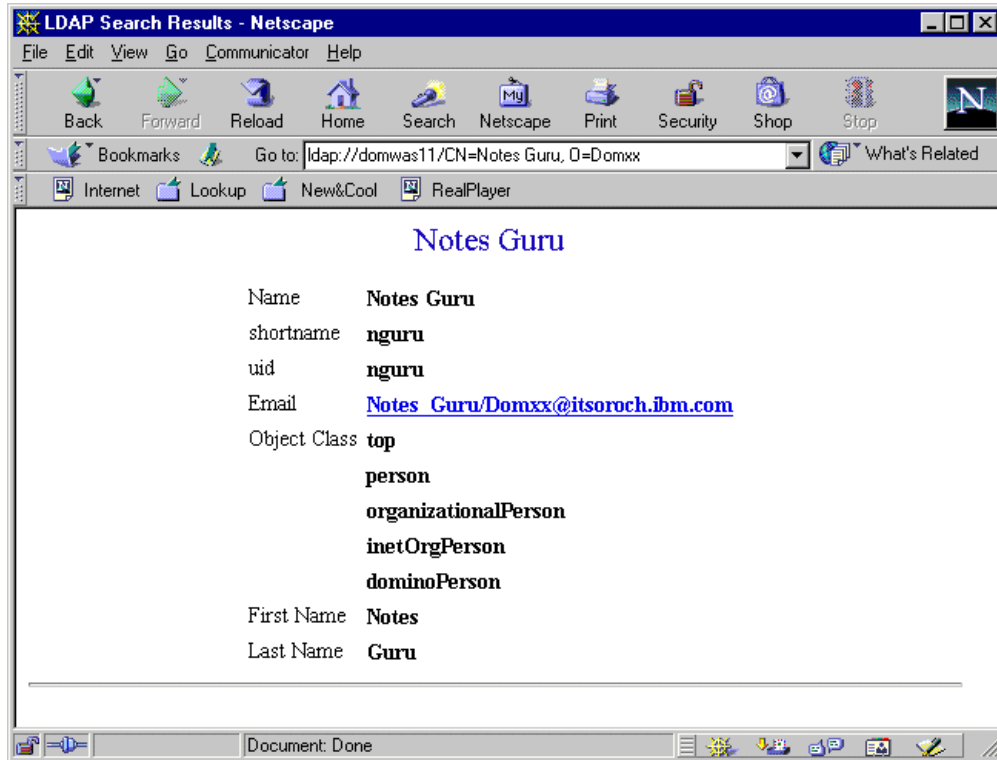


Figure 4-44 Netscape LDAP directory lookup

### Using the Qshell utility

To verify a connection to the LDAP server using the Qshell utility, perform the following steps:

1. Open a 5250 session to your iSeries server and enter the following OS/400 command  
STRQSH or QSH  
to start the Qshell Interpreter.
2. To search and display all directory entries in the Domino Directory in LDAP format, enter the following command:  

```
ldapsearch -H DOMWASxx -p389xx objectclass=*
```

  
Here DOMWASxx is your Domino server name and 389xx is your Domino LDAP server port if it is different from the default port of 389.

## 4.5 Configuring WebSphere to use Domino LDAP

This section describes configuring WebSphere to use the Domino LDAP server for user authentication. To complete the examples in this section you should complete the configuration of Domino as an LDAP server as described in Section 4.4.1, "Configuring the Domino Directory for LDAP access" on page 95.

Similar to Domino, WebSphere Application Server does not by default require users to enter a user name and password to access any of the resources it manages. Several steps need to be performed to activate security. With WebSphere Application Server, you can secure your applications by configuring the following security policies:

- ▶ **Authentication:** Determination of who is making a request
- ▶ **Authorization:** Determination of whether a request is to be honored
- ▶ **Delegation:** Determination of who will be handling the request

To restrict the use of the WebSphere Application Server by allowing only *authenticated users* access to its resources, a user registry (a directory of valid users) needs to be in place. This can either be the user registry provided by the platform on which WebSphere is running (for OS/400 this would be the user profiles), or any LDAP server. On an iSeries server, an LDAP server can be either SecureWay Directory for OS/400 or Lotus Domino for iSeries.

If you plan to use the same user registry for multiple servers and later enable single sign-on, you *must* use an LDAP server for authentication.

### 4.5.1 Authentication concepts of the WebSphere Application Server

Authentication policy determines how authentication is accomplished. Authentication is performed in two steps:

- ▶ Acquiring the authentication data of a principal.
- ▶ Validating the authentication data against a user registry.

An authentication mechanism validates authentication data against an associated user registry. WebSphere Application Server offers these choices for authentication mechanisms:

- ▶ Lightweight Third Party Authentication (LTPA)
- ▶ Native OS

#### ***Lightweight Third Party Authentication (LTPA)***

LTPA uses a trusted third-party server to authenticate the user. This is the heart of the distributed security model. Since all application servers trust the third-party authentication server, security information can be passed along with requests from one application server to another. Use the LTPA authentication mechanism when applications are distributed across multiple application servers.

#### ***Native OS***

The native OS authentication mechanism uses native OS/400 routines (OS/400 user profiles) to authenticate the user. Native OS is easier to configure than LTPA, but can be used for only the simplest topologies. Note that authenticating through the native OS mechanism *does not* log the user onto the iSeries server. Even though user profiles and passwords are used for authentication, no jobs or threads are executed under the users' profiles.

A challenge type specifies how a server will challenge and retrieve authentication data from a user. The choices for challenge type are:

- ▶ **None:** The user is not challenged for authentication data. If the requested resource is protected, then the user will not be served the resource.
- ▶ **Basic:** The user is challenged for a user ID and password.
- ▶ **Custom:** Applicable only to Web clients. Custom challenge type is used when one wants to configure the server to use a customized HTML form to retrieve the user ID and password.
- ▶ **Certificate** (new with v3.5): Applicable only to Web clients. The user is required to present a digital certificate (X.509) to establish the connection. With certificate challenge type, the Web server is trusted to authenticate the user through the SSL exchange. Then for authorization purposes, the WebSphere security infrastructure identifies the principal by extracting information from the certificate and mapping it to an entry in the user registry.

A user registry is where the user and group information is stored. It contains a mapping of principals to authentication information and privilege attributes such as access ID, password and group IDs. A “principal” is a representation of a human user or system entity such as a server process. The choices for user registry are:

- ▶ Native OS - OS/400 profiles
- ▶ Lightweight Directory Access Protocol (LDAP)

## 4.5.2 Enabling global security in the WebSphere Application Server

In this section you will learn how to enable security for your WebSphere Application Server using Domino LDAP in order to authenticate users before giving them access to resources.

To configure the WebSphere Application Server to use the Domino LDAP server, you must enable global security. This is done via the WebSphere Administrative Console. To enable global security and configure WebSphere to use the Domino LDAP server, perform the following steps:

1. From your administrator PC workstation, start the WebSphere Administration Console.
2. Click the **Wizard** icon and select **Configure Global Security Settings** as shown in Figure 4-45.

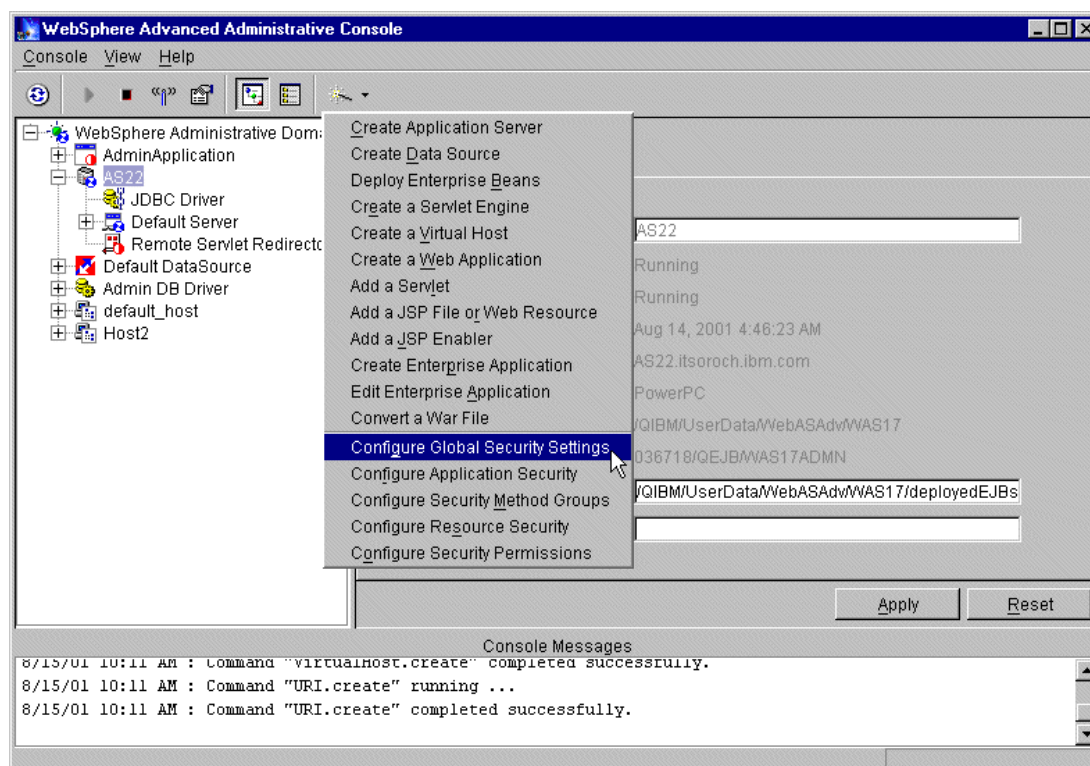


Figure 4-45 Configuring global security

3. The Set Global Security wizard starts. On the first pane of the General tab, check **Enable Security**. Verify that the Security Cache Timeout is set to a reasonable value. When the timeout is reached, WebSphere Application Server clears the security cache and rebuilds the security data. If the value is set too low, the extra processing overhead may be unacceptable. If the value is set too high, you create a security risk by caching security data for a long period of time. The default value is 600 seconds. See Figure 4-46. Click **Next**.

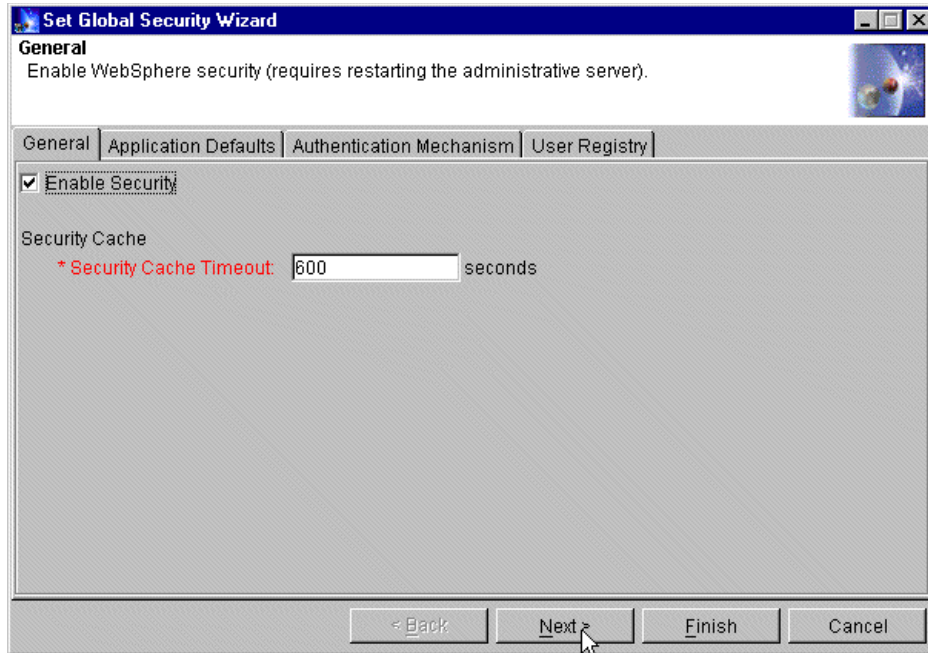


Figure 4-46 Set Global Security Wizard - General tab

4. The Application Defaults tab is displayed. Set the Realm Name field to your domain portion of your fully qualified Internet name for the system running your WebSphere administrative domain. In our example the realm is `itsoroch.ibm.com`. This means security will apply to machines in this realm such as `as22.itsoroch.ibm.com`.

For the Challenge Type, choose **Basic (User ID and Password)** as shown in Figure 4-47. Click **Next**.

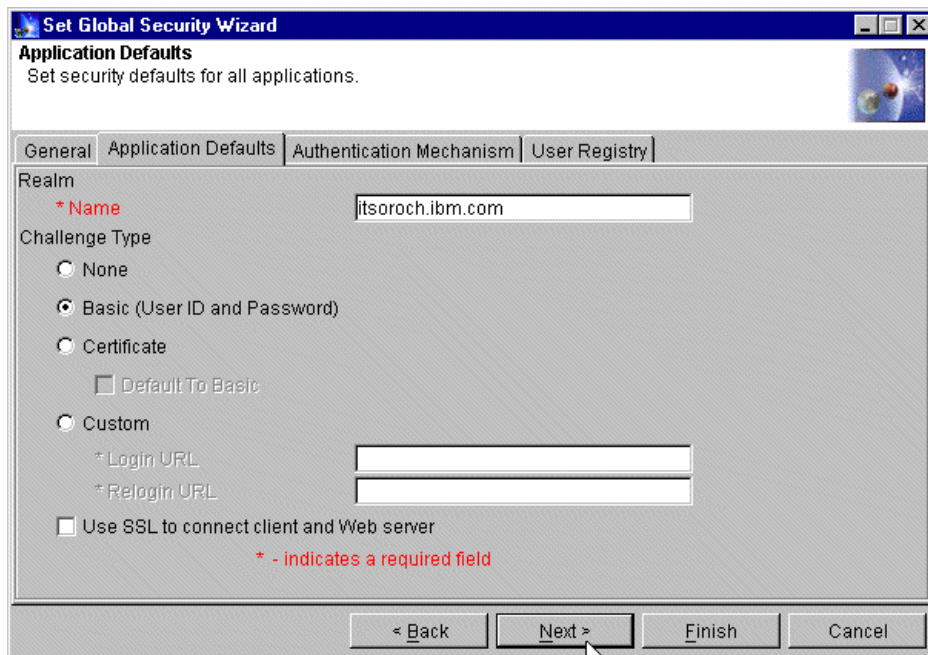


Figure 4-47 Set Global Security Wizard - Application Defaults tab

5. The Authentication Mechanism tab is displayed. Select **Lightweight Third Party Authentication (LTPA)**, as shown in Figure 4-48. Click **Next**.

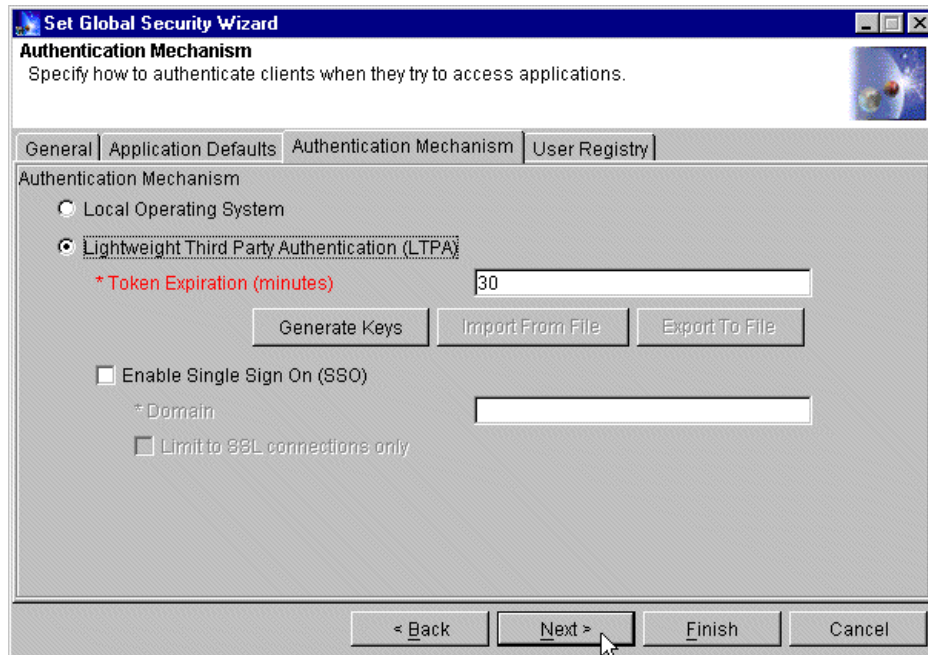


Figure 4-48 Set Global Security Wizard - Authentication Mechanism tab

6. The User Registry tab is displayed. This is where you tell WebSphere that you want to use Domino R5.0 as the user registry. WebSphere will then use LDAP to connect to the Domino server and authenticate users. The Domino LDAP server task must be running as the wizard will attempt to connect to the LDAP server to verify the information you enter. Fill in the fields as explained the following list:

– **Security Server ID**

This is the user ID of the Administrator for the WebSphere administrative domain. This user ID is used later when accessing WebSphere administration services using the WebSphere Administrative Console. By default, this should be the value of the short name or user ID for a user already defined in the LDAP directory. Do not specify a Distinguished Name by using cn= or uid= before the value.

**Note:** Later, when you restart the WebSphere Administrative Console, you must enter the value exactly as you specified it in this field.

– **Security Server Password**

This is the valid password for the Security Server ID. This field is case sensitive.

– **Directory Type = Domino 5.0**

This value should be set for the type of LDAP server you are using. For example, select SecureWay for IBM SecureWay LDAP directory, or Domino 5.0 for the Domino LDAP directory.

– **Host**

This is the fully qualified host name on which the LDAP directory is running.

– **Port**

This is the port on which the LDAP directory runs. You may leave this field blank for the default, non-SSL (Secure Sockets Layer) port of the LDAP directory (port 389).

– **Base Distinguished Name**

This is the Distinguished Name (DN) of the directory in which searches begin within the LDAP directory. For example, for a user with a DN of cn=John Doe, ou=Rochester, o=IBM, and c=US, you could specify a base DN of ou=Rochester, o=IBM, c=US or o=IBM, c=us or c=us. This is a required field for all LDAP directories except the Domino Directory.

**Note:** If you are using the Domino Directory, and you specify a Base Distinguished Name, you cannot grant permissions to individual Web users for resources managed by your WebSphere application server.

– **Bind Distinguished Name**

This is the DN of the user who is capable of performing searches on the directory. In most cases, this field is not required since all users are usually authorized to search an LDAP directory. However, if the LDAP directory contents are protected from all LDAP users, you need to specify the DN of an authorized user, such as the administrator of the directory (for example, cn=Administrator).

– **Bind Password**

This is the valid password for the user specified as the Bind Distinguished Name. This is required only if you specify a value for Bind Distinguished Name. This field is case sensitive.

Complete the fields as shown in Figure 4-49. At a minimum, you will need to change the Security Server ID, Security Server Password, Directory Type and Host fields to match your environment.

Click **Finish** to save the Global Security Settings.

The screenshot shows the 'Set Global Security Wizard' window, specifically the 'User Registry' tab. The window title is 'Set Global Security Wizard' and the subtitle is 'User Registry'. Below the subtitle is the instruction 'Specify details about the Authentication Mechanism you chose.' The window has four tabs: 'General', 'Application Defaults', 'Authentication Mechanism', and 'User Registry'. The 'User Registry' tab is selected. The main area is titled 'LDAP' and contains several fields: '\* Security Server ID' (text box with 'nguru'), '\* Security Server Password' (password box with '\*\*\*\*\*'), '\* Directory Type' (dropdown menu with 'Domino 5.0'), '\* Host' (text box with 'domwas11'), 'Port' (text box), 'Base Distinguished Name' (text box), 'Bind Distinguished Name' (text box), and 'Bind Password' (text box). There is an 'Advanced...' button to the right of the 'Directory Type' dropdown. At the bottom left, there is a checkbox labeled 'Use SSL to connect to directory' which is unchecked. A red asterisk is followed by the text '- indicates a required field'. At the bottom of the window are four buttons: '< Back', 'NEXT >', 'Finish', and 'Cancel'. The 'Finish' button is highlighted with a mouse cursor.

Figure 4-49 Set Global Security Wizard - User Registry tab



7. You are then prompted for an LTPA password as shown in Figure 4-50. This is the password for the keys that WebSphere is generating. Enter a password and click **OK**.

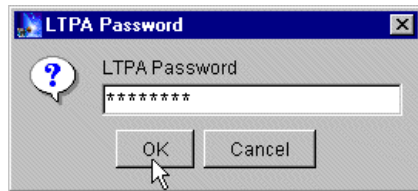


Figure 4-50 LTPA Password

8. After a couple of seconds, a message box appears (see Figure 4-51) stating that your changes will not take effect until the WebSphere Administrative server is restarted. Click **OK** to continue.

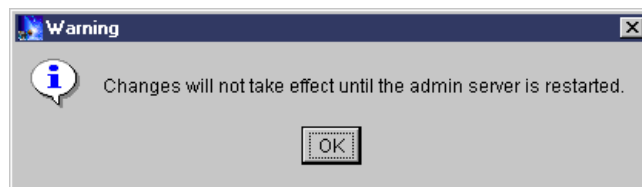


Figure 4-51 Global Security Settings finished successfully message

9. To restart the WebSphere Administrative server, right-click the node and select **Restart** (see Figure 4-52).

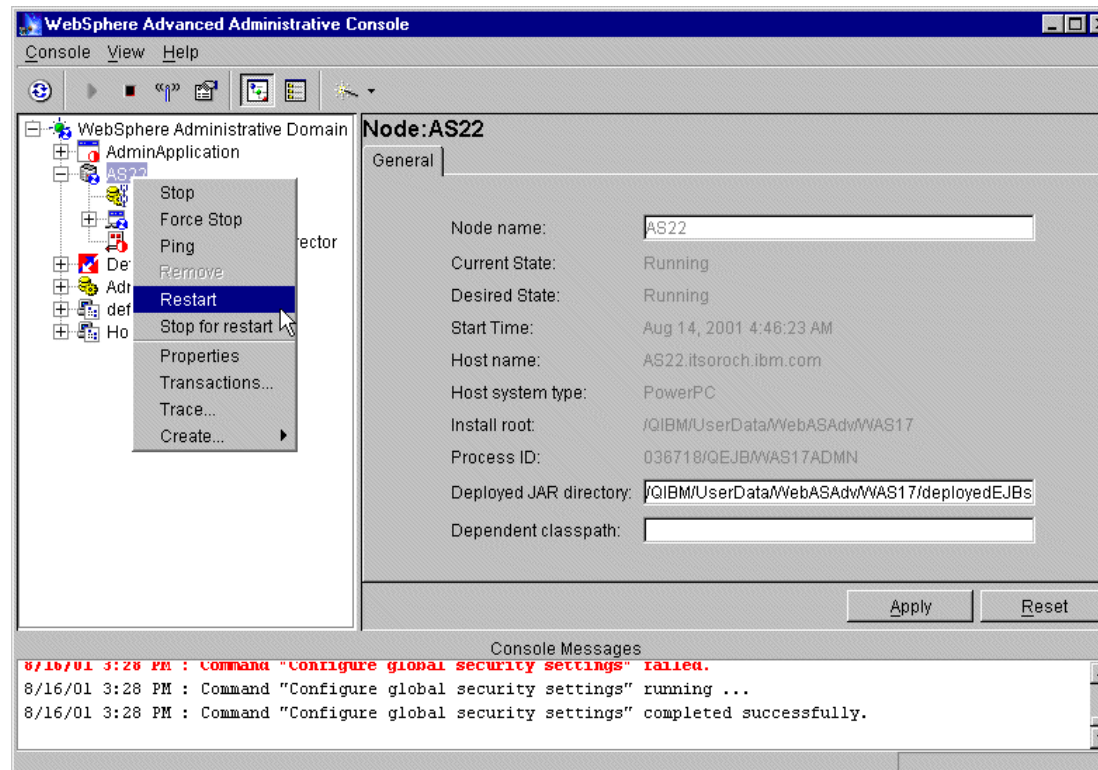


Figure 4-52 Restarting the WebSphere Administration server

10. A warning message appears (see Figure 4-53) stating that you are trying to stop the node to which the console is connected. Click **Yes** to continue.

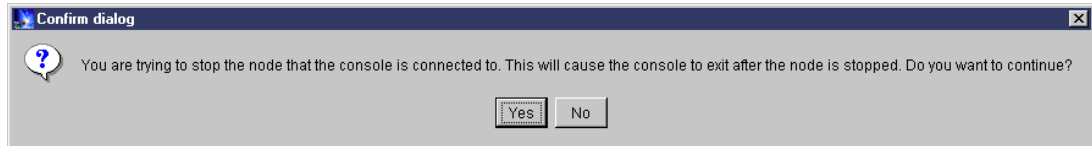


Figure 4-53 Restart WebSphere warning message

11. Make sure that your WebSphere monitor job (WASxxMNTR) and the administrative server job (WASxxADMN), where WASxx is the name of your WebSphere Application Server instance, is completely restarted before continuing. To check this, perform the following tasks:

- a. Enter the Work with Active Jobs (WRKACTJOB) command to verify that these jobs are in JVAW and EVTW status:

```
wrkactjob sbs(qejbsbs)
```

Monitor the administrative server task (or job) (WASxxADMN) from the Work with Active Jobs window (WRKACTJOB) to ensure that the WebSphere Administrative server restarts successfully. As you watch the Administrative server job, notice that it stops, starts, stops, and then starts again. This is expected after Global Security Settings have been enabled or changed.

- b. Display the job log for the administrative server job (WASxxADMN). From the WRKACTJOB display, enter option 5 (Work with) next to the WASxxADMN job. Press Enter. Note, it may take several minutes - possibly more than 10 minutes - depending on your system.
- c. On the Work with Job display, enter option 10 (Display job log, if active or on job queue) to display the job log. Press Enter.
- d. Look for the following message:

```
WebSphere administration server WASxxADMN ready.
```

You may have to press F5 several times to refresh the display and scroll down if you see More... in the lower right corner of your display.

- e. Position the cursor on the ready message (step d). Press F1 to verify that the administration server is listening to the correct port assigned to your team (9xx) and that the message is new (look at the time stamp). See Figure 4-54.

Additional Message Information			
Message ID . . . . .	EJB0106	Severity . . . . .	00
Message type . . . . .	Information		
Date sent . . . . .	02/18/01	Time sent . . . . .	11:13:53
Message . . . . . : WebSphere administration server WASxxADMN ready.			
Cause . . . . . : WebSphere administration server WASxxADMN is ready to handle requests from the WebSphere administration console on port 9xx.			
Diagnostic information is controlled on port 20834.			

Figure 4-54 WebSphere administration server started successfully message

12. Once the WebSphere Administrative Server has been successfully and completely restarted, restart the WebSphere administrative console and connect it to your WebSphere instance.



13. Notice that you now need to sign on to the WebSphere Administrative console because you have enabled security. Specify the user ID and password exactly as you configured them previously for the Security Server ID and Security Server Password fields in the Global Security Settings wizard (see Figure 4-55).

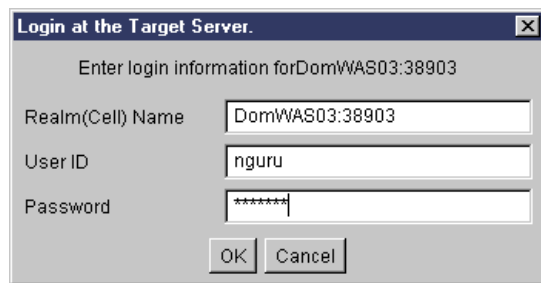


Figure 4-55 Sign on to the WebSphere Administration Console

14. Wait until the WebSphere Administrative message console displays the Console Ready message before continuing.

This completes the configuration of WebSphere to use the Domino Directory as an LDAP server. In order to test this, you need to apply WebSphere security to some WebSphere components. See Section 4.6, “Securing WebSphere resources” on page 109, for an example of how you can secure WebSphere resources.

## 4.6 Securing WebSphere resources

After activating global security for your WebSphere Application Server instance, you need to protect your resources. This is called “creating an authorization policy”, and is performed from the WebSphere Administrative Console. This section provides an example of how you can secure WebSphere resources using the following steps:

- ▶ “Creating an enterprise application” on page 109
- ▶ “Configuring application security” on page 112
- ▶ “Configuring resource security” on page 115
- ▶ “Configuring security permissions” on page 118
- ▶ “Starting the enterprise application” on page 121

The following sections describe these steps in detail. To complete the examples in this section, you must first enable global security for your WebSphere Application Server instance as described in Section 4.3.2, “Enabling global security in the WebSphere Application Server” on page 81 using OS/400 LDAP or in Section 4.5.2, “Enabling global security in the WebSphere Application Server” on page 103 using Domino LDAP.

### 4.6.1 Creating an enterprise application

To protect any resources managed by WebSphere, an enterprise application needs to be created. An Enterprise Application is a collection of resources that can be protected as a whole. It is usually comprised of enterprise beans, servlets, JavaServer Pages, and other resources that perform an aspect of business logic.

1. From the WebSphere Administrative Console, click the **Wizard** icon as shown in Figure 4-56, and select **Create Enterprise Application**. Or, from the WebSphere Administrative Console pull-down menu, select **Console -> Tasks -> Create Enterprise Application**.

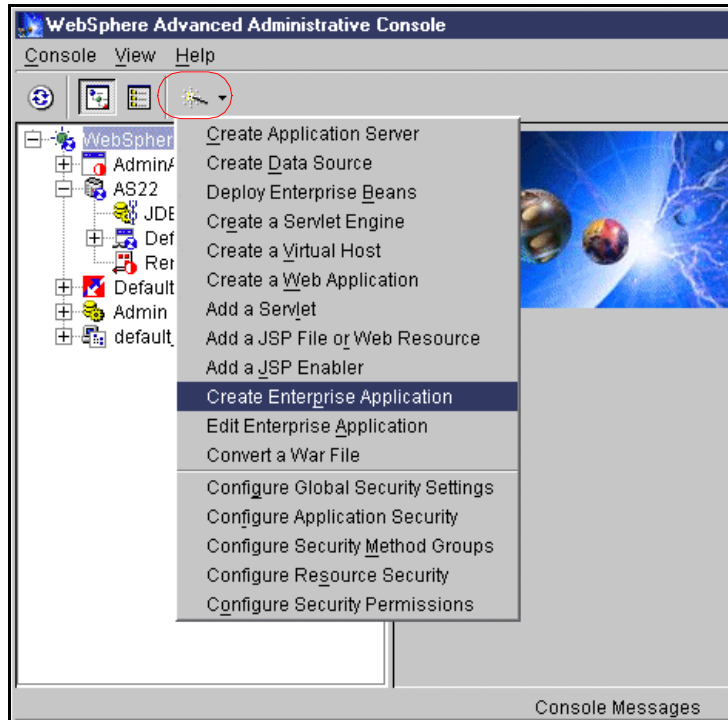


Figure 4-56 Create Enterprise Application

2. On the Application Details window (see Figure 4-57), enter the name for the new Enterprise Application. In our example we specified an Application Name of SimpleEA. Click **Next** to continue. Make sure not to click the Next button twice. It may take several minutes until the next window appears. Also the busy icon (hourglass) will not appear unless you move the cursor off the Create Enterprise Application Wizard window.

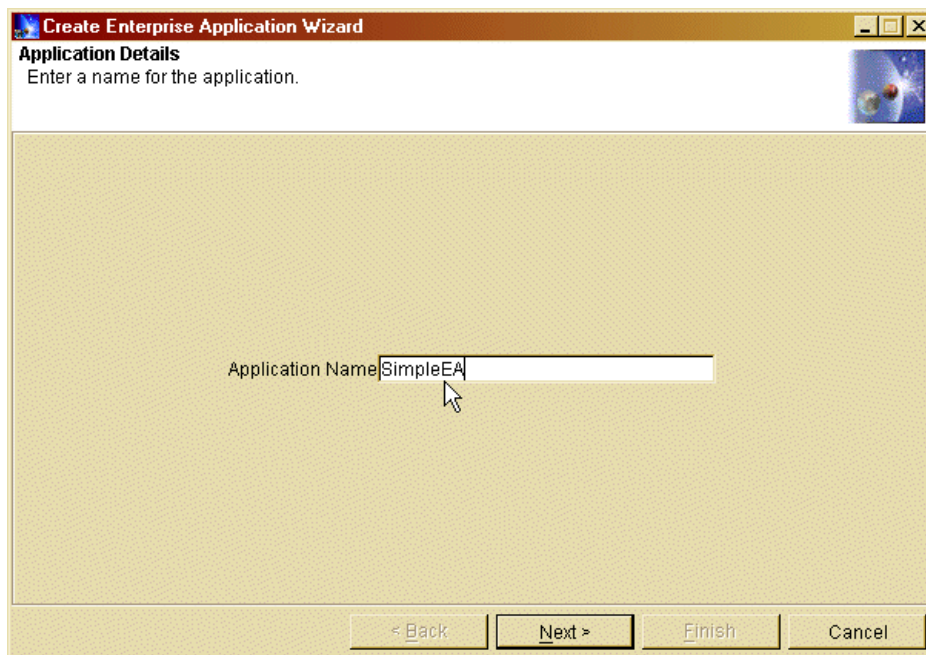


Figure 4-57 Create Enterprise Application Wizard: Application Details

3. After several minutes, the Application Resources window (see Figure 4-58) appears. Once you see it, click the plus (+) sign next to Web Applications to expand the tree underneath it. It may also take several minutes to expand the tree.
4. Select your Web application, and click the **Add** button. In our example we selected a Web application by the name of *DomApp*. Click the **Next** button.

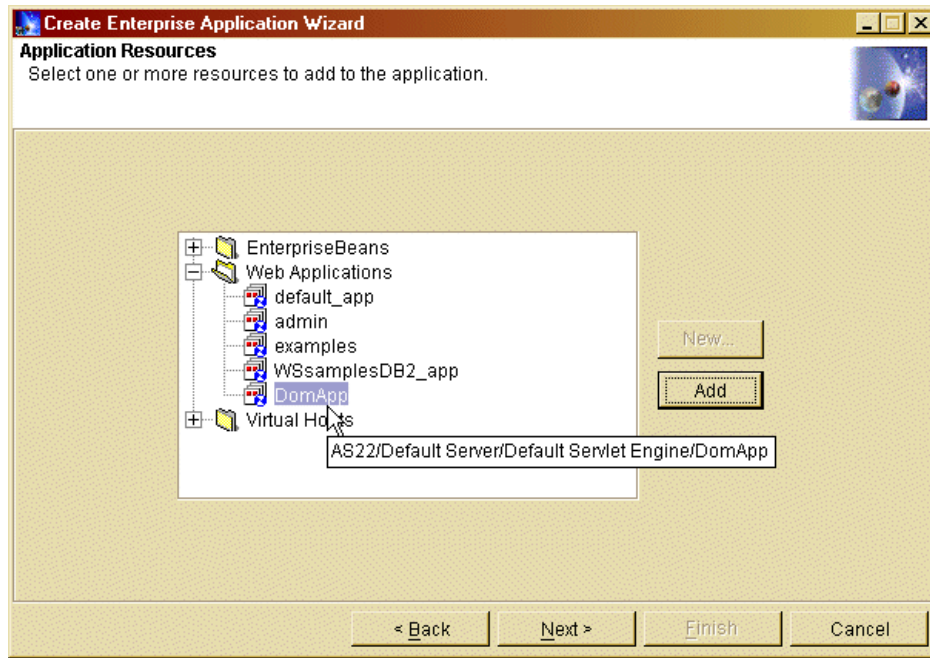


Figure 4-58 Create Enterprise Application Wizard: Application Resources

5. You now see another Application Resources window (Figure 4-59). This allows you to remove unwanted Web applications from your new Enterprise Application. Click the plus (+) sign to expand Web Applications and ensure your Web Application is added. Again in our example this is the Web Application of DomApp. Click **Finish**.

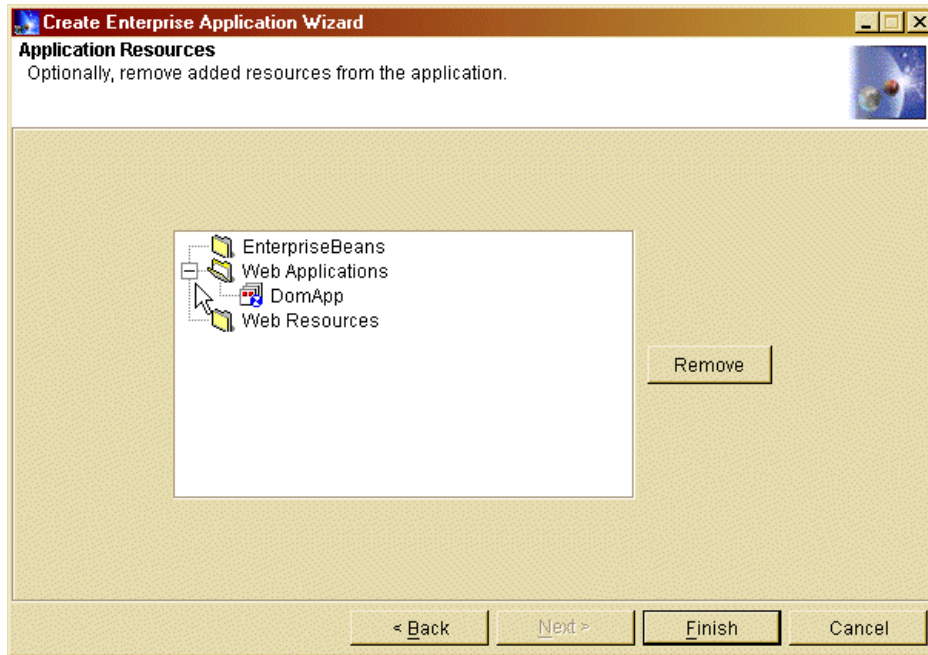


Figure 4-59 Create Enterprise Application Wizard: Application Resources - Remove resources

6. It may take a few of minutes to create the Enterprise Application. Wait until the confirmation window is shown (Figure 4-60). Click **OK**.

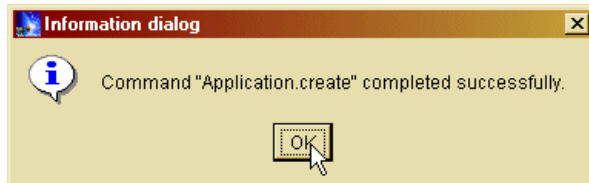


Figure 4-60 Enterprise Application created successfully message

A new Enterprise Application has now been created within your WebSphere Application Server.

## 4.6.2 Configuring application security

In this section, you enable security for your Enterprise Application. Perform the following steps:

1. From the WebSphere Administrative Console, click the **Wizard** icon and select **Configure Application Security** as shown in Figure 4-61.



Figure 4-61 Configure Application Security

2. On the Enterprise Applications window, click the plus (+) sign next to Enterprise Applications to expand the tree. In our example, this was **SimpleEA**. Click **Next** (see Figure 4-62).

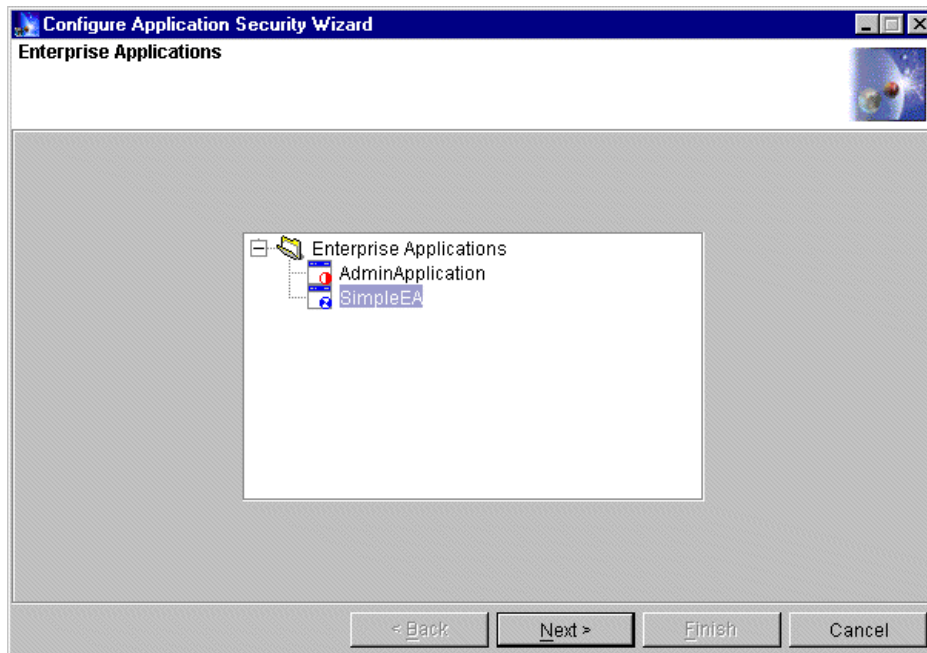


Figure 4-62 Configure Application Security Wizard: Enterprise Applications

3. On the Realm and Challenge Type window, enter your Realm Name and select the Challenge Type of **Basic (User ID and Password)**. Leave the rest of the options as the defaults. In our example, the realm name is *ITSORoch.IBM.COM*. Click **Next** (see Figure 4-63).

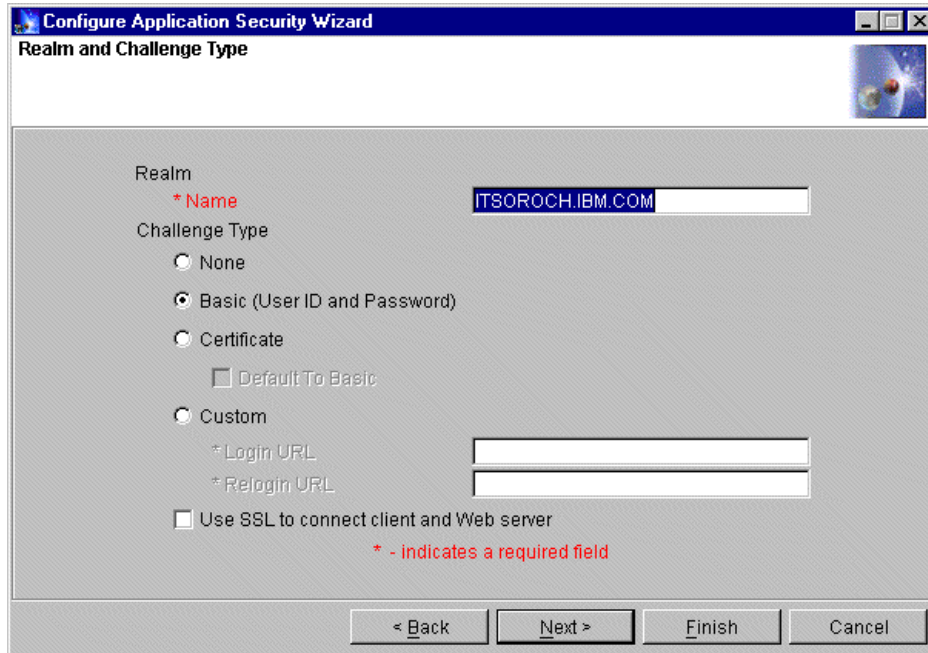


Figure 4-63 Configure Application Security Wizard: Changing Realm and Challenge Type

4. Leave the Application Identity window defaults, and click **Finish** (see Figure 4-64).



Figure 4-64 Configure Application Security Wizard: Application Identity

5. After the console message appears stating that Application Security was configured successfully (see Figure 4-65), you can continue.



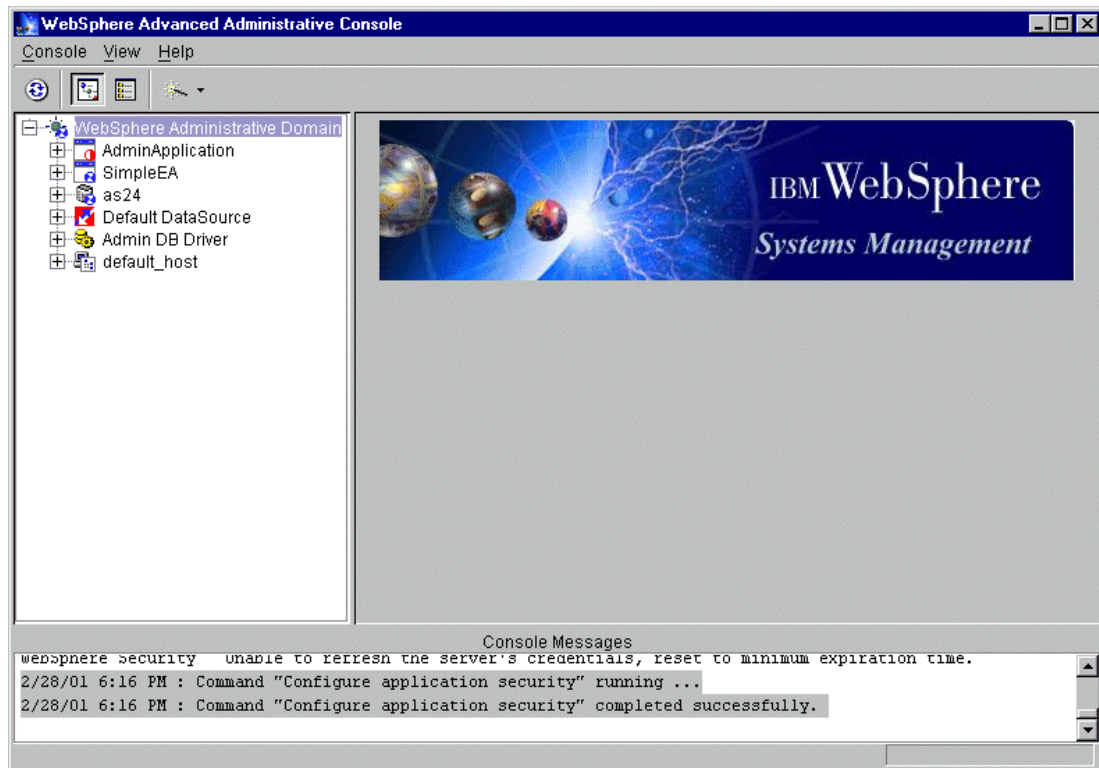


Figure 4-65 Application Security configured successfully

### 4.6.3 Configuring resource security

To enable security for the Enterprise Application, you must configure resource security. This sets security for the components of the Enterprise Application. Perform the following steps:

1. From the WebSphere Administrative Console, click the **Wizard** icon and select **Configure Resource Security** as shown in Figure 4-66.

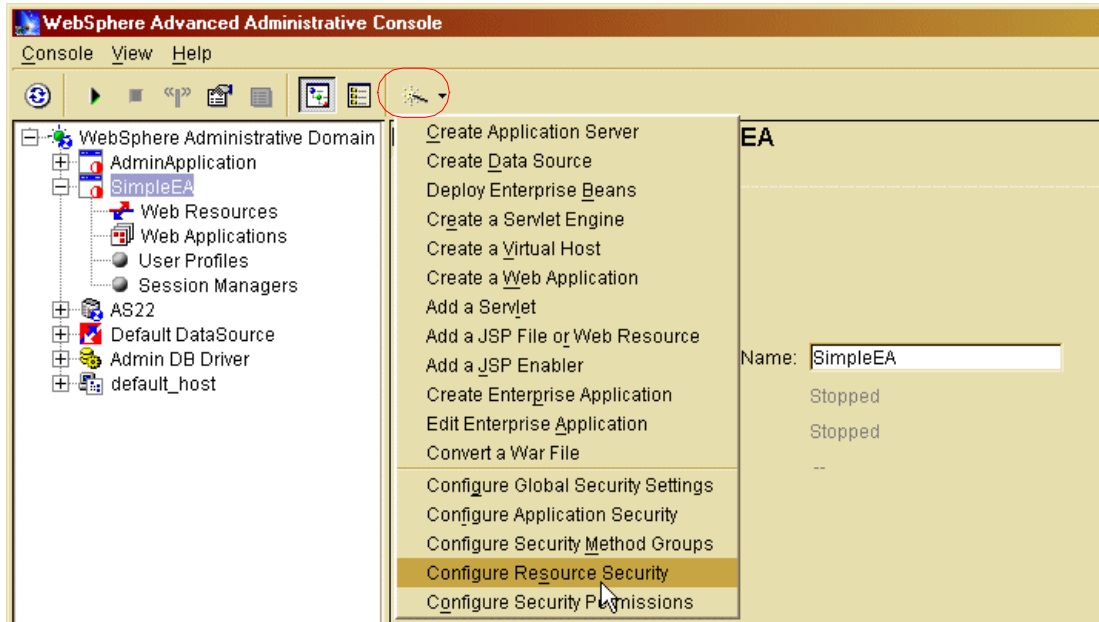


Figure 4-66 Configure Resource Security

2. Once the Configure Resource Security Wizard window appears, expand the Virtual Hosts tree by clicking the plus (+) sign next to Virtual Hosts.

**Note:** Be patient and click the plus sign only once because it takes a few minutes until the tree expands. When you perform this step the first time, the mouse pointer may not change to an hourglass.

3. Expand the default\_host tree. Again, it may take a while to expand the tree for the very first time (see Figure 4-67).



Figure 4-67 Configure Resource Security Wizard: Resources



4. The default\_host tree name is longer than what can be shown in the dialog pane. Scroll down and select your resource to be secured. In our example we selected **/webapp/DomApp/SimpleServlet**. Click **Next** (see Figure 4-68).

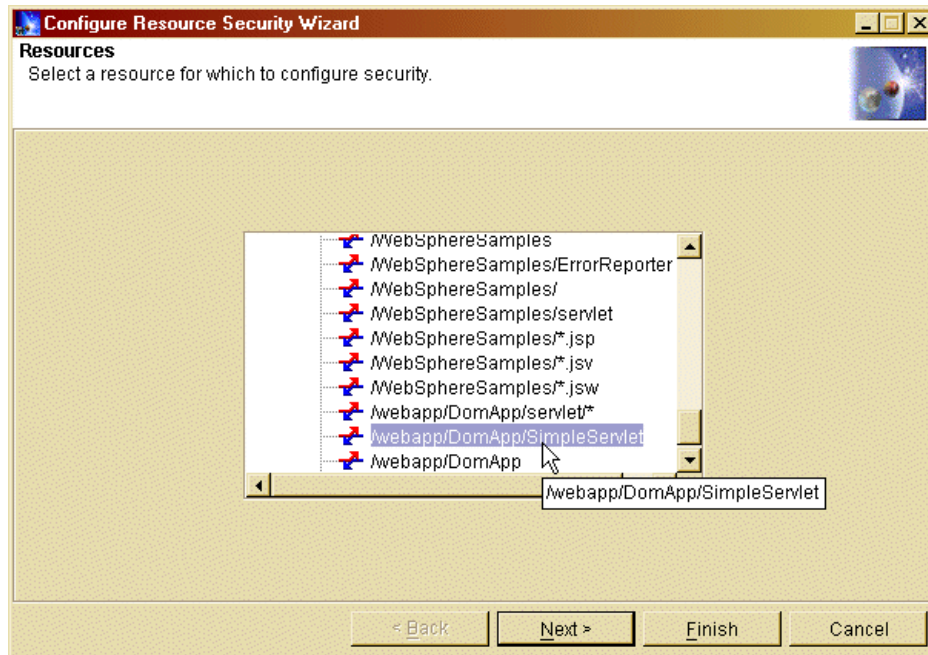


Figure 4-68 Configure Resource Security Wizard: Scrolling through resources

5. A message appears asking to use the default method groups since no method groups have been assigned yet. Click **Yes** (see Figure 4-69).

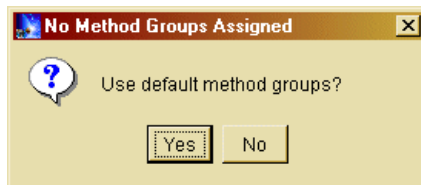


Figure 4-69 Configuring Resource Security to use default method groups

6. On the Method Groups window, click **Finish** (see Figure 4-70).

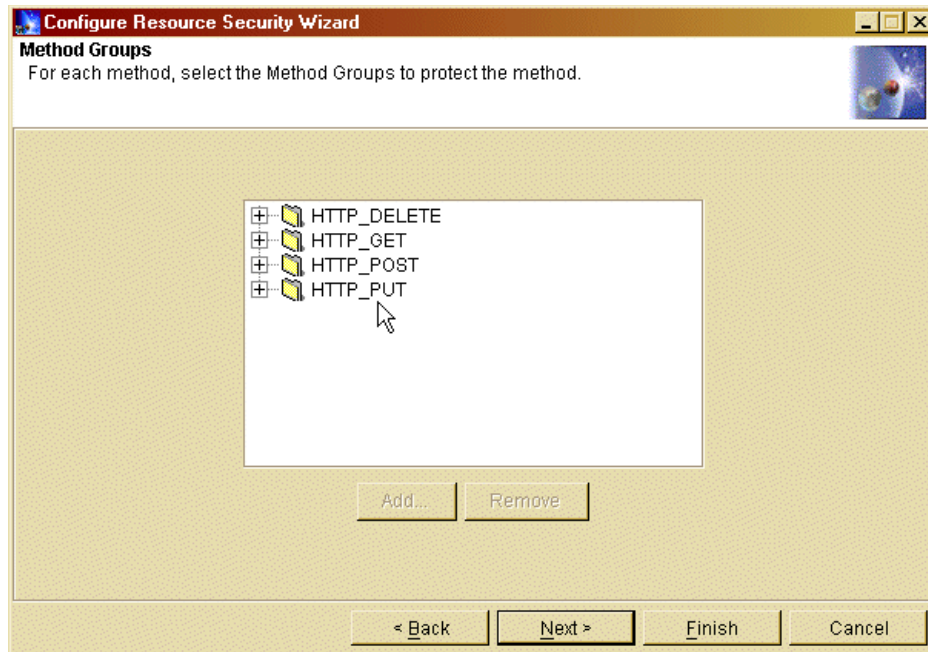


Figure 4-70 Configure Resource Security Wizard: Method Groups

At this point, you have protected the methods of your Web application. You still must define which users are allowed to access them. This is done by configuring Security Permissions.

#### 4.6.4 Configuring security permissions

To assign certain users or groups access to different methods and applications, perform the following steps:

1. From the WebSphere Administrative Console, click the **Wizard** icon and select **Configure Security Permissions** (see Figure 4-71).

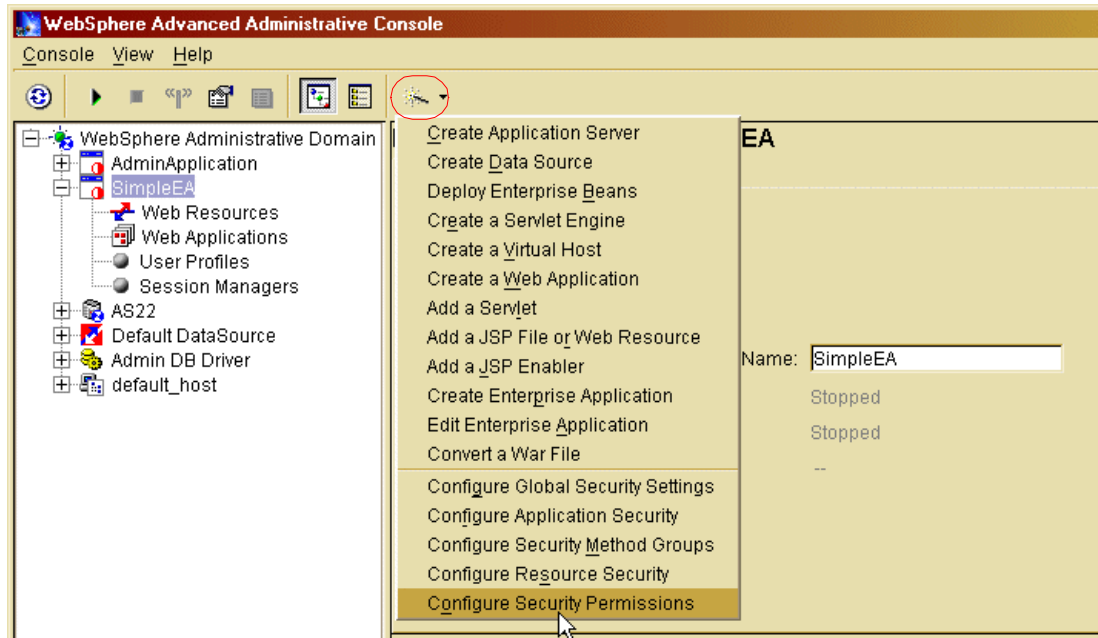


Figure 4-71 Configure Security Permissions

2. On the Enterprise Applications window, expand the tree by clicking the plus (+) sign under Enterprise Applications.
3. Select your Enterprise Application (in our example this is **SimpleEA**) and click **Next** (see Figure 4-72).

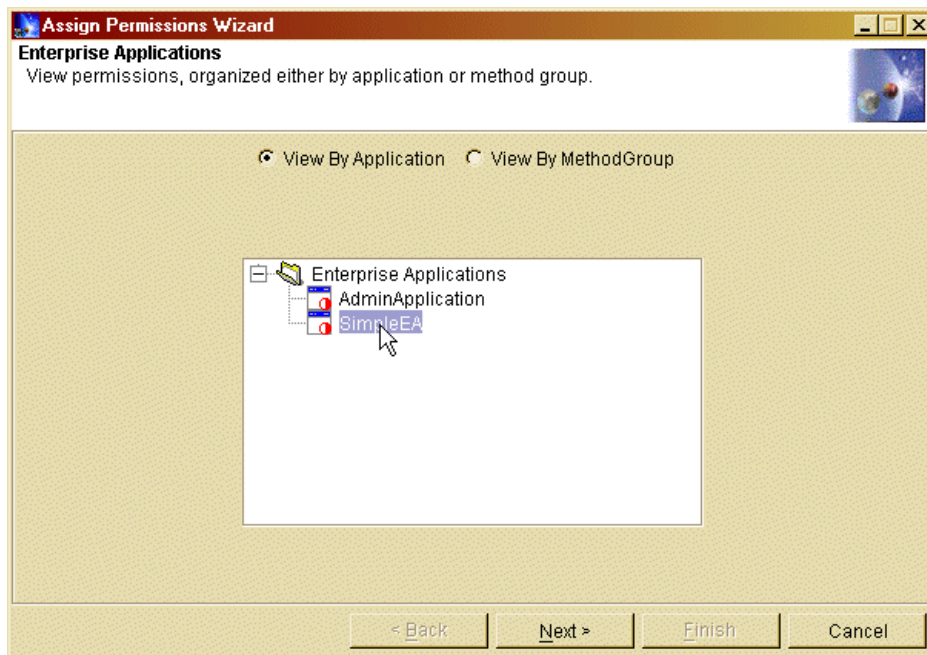


Figure 4-72 Assign Permissions Wizard: Enterprise Applications

4. On the Permissions window (see Figure 4-73), select all the methods. To do so, click the first method, hold down the Shift key, and click the last method. Click **Next**.

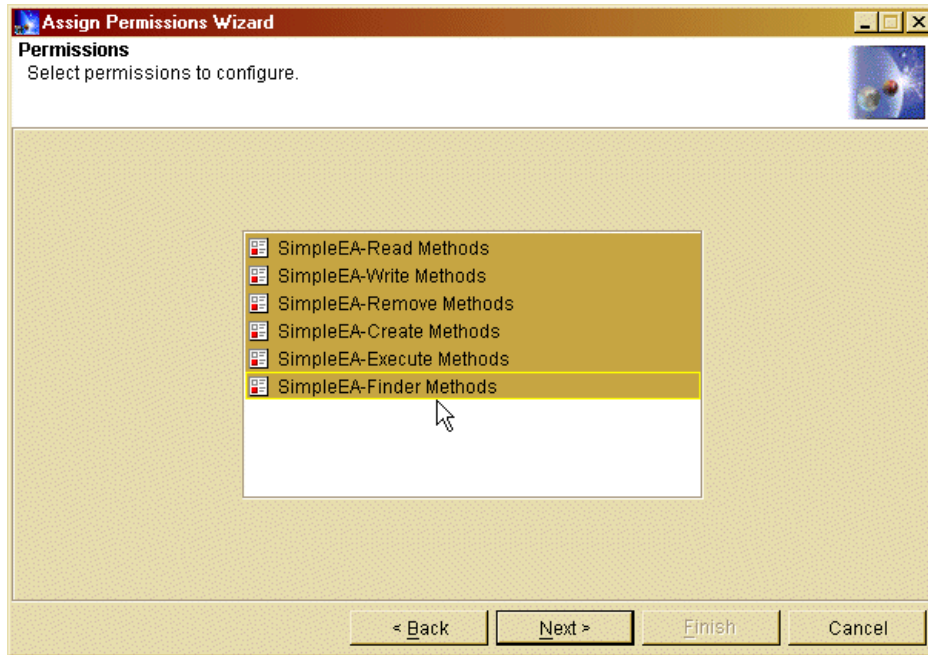


Figure 4-73 Assign Permissions Wizard: Selecting Permissions to configure

5. On the Grant Permissions window, click **All Authenticated Users** (see Figure 4-74).

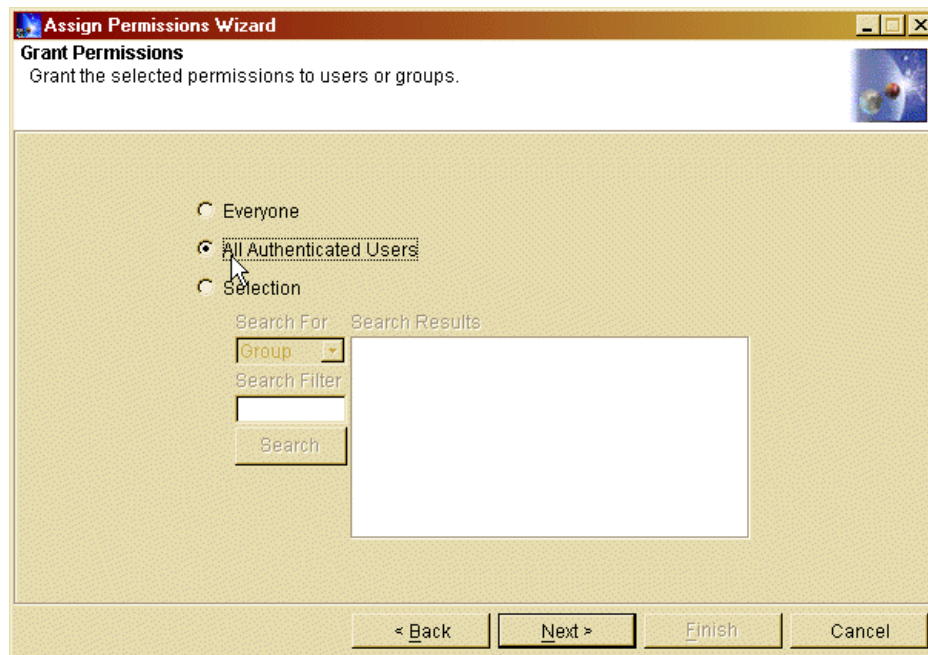


Figure 4-74 Assign Permissions Wizard: Grant Permissions

**Note:** With WebSphere Application Server and Lotus Domino 5.0.6a, single sign-on does not allow you to protect resources for individual users. You can select either **All Authenticated Users** (as in this example), or you can grant authority to a group.

The window shown in Figure 4-74 can also be used to verify whether you have access to the correct entries in the LDAP directory. To do this, click the **Selection** radio button, select **User** in the Search For pull-down menu, type \* in the Search Filter field, and click **Search**.

6. On the Remove Permissions window (see Figure 4-75), click **Finish** to continue.

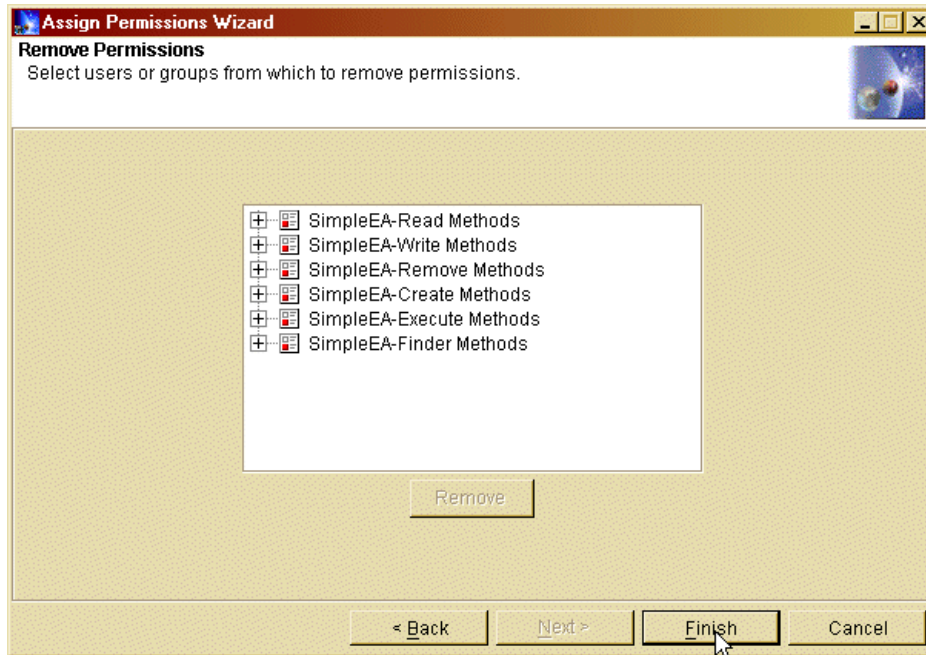


Figure 4-75 Assign Permissions Wizard: Remove Permissions

7. The following message in the Console Messages pane appears:  
Command "Configure permissions" completed successfully.

## 4.6.5 Starting the enterprise application

To activate the security settings you configured in the previous sections, the enterprise application must be started. To do this, perform the following steps:

1. From the WebSphere Administrative Console, right-click your enterprise application and select **Start** (see Figure 4-76). In our example this was **SimpleEA**.

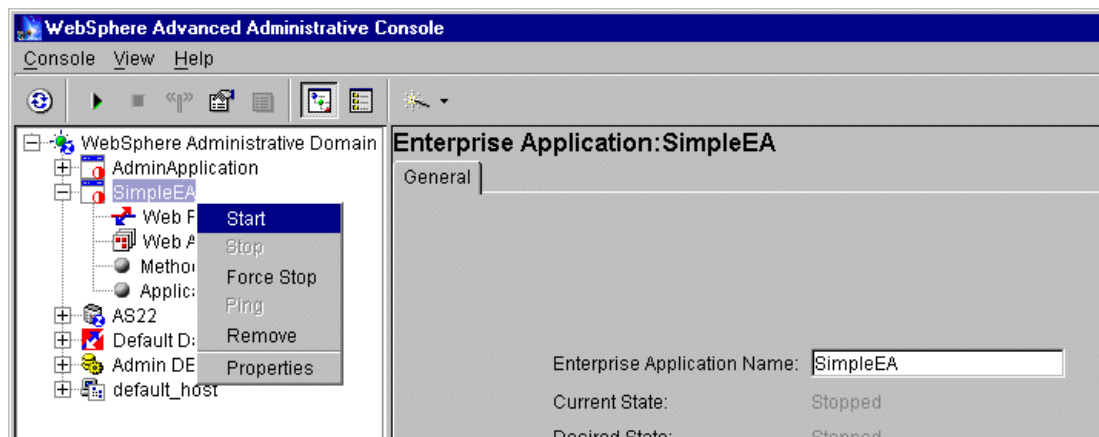
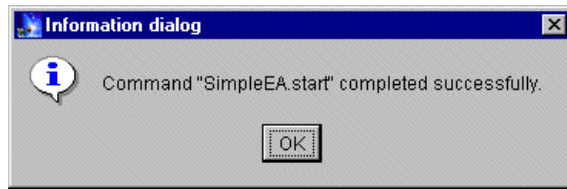


Figure 4-76 Start Enterprise Application

2. It takes several minutes until the successful completion message appears. Click **OK** (see Figure 4-77).



*Figure 4-77 Start Enterprise Application completed successfully*



## Single sign-on

For many Web applications, security is a main concern. Both WebSphere Application Server and Domino provide strong support for securing application access and data. Both products implement security mechanisms that involve determining and verifying user identity (authentication) and allowing access to protected resources only to designated users (authorization).

However, the WebSphere and Domino security architectures are different. For example, WebSphere's access control model is based on capabilities associated with a user; these define what a user can do (read, write, execute, and so on) with specified objects. Domino's access control model is resource-based, limiting actions users can perform according to entries in the Access Control List (ACL) stored with the database. It is further refined by document- and field-level access controls that can refine the basic ACL controls. Providing security for an application supported across both products requires establishing user identity and access controls in both products.

One aspect of the Domino and WebSphere security models, the directory (referred to as the "user registry" in WebSphere), can be made common. This is possible since both Domino and WebSphere support the Lightweight Directory Access Protocol (LDAP) for directory access. The Domino Directory component provides for LDAP access and WebSphere can be set up to use an LDAP-accessible directory as its user registry.

Once a common user registry has been created by configuring Domino and WebSphere to rely on the same LDAP server for authentication, users will find it tedious to enter the same user ID and password multiple times, depending on whether they need to access a Domino object (a Domino database, a view or a document) or a WebSphere resource, such as an HTML file, a servlet, JavaServer Pages (JSP) or an Enterprise JavaBean (EJB). To solve the problem of signing on multiple times, a new function called single sign-on (SSO) can be implemented in Lotus Domino as well as on the WebSphere Application Server.

This chapter contains information on configuring Web single sign-on for Lotus Domino and the WebSphere Application Server.

## 5.1 What is single sign-on?

Using single sign-on, Web users can authenticate once to a Domino server or to a WebSphere Application Server, and then access any other Domino servers or WebSphere Application Servers in the same DNS domain that are enabled for SSO without logging on again. This is accomplished by configuring the Domino servers and the WebSphere Application Servers to share authentication information gathered from a single LDAP server.

## 5.2 SSO prerequisites

Prerequisites associated with SSO support for Domino servers and WebSphere Application Servers are as follows:

- ▶ All servers must be configured for the same DNS domain. For example, if the DNS domain is specified to be "mycompany.com", then SSO will be effective with any Domino or WebSphere Application Server that serves the "mycompany.com" domain such as "a.mycompany.com" and "b.mycompany.com".
- ▶ All servers must share the same user registry, accessible using LDAP. A Domino Directory (configured for LDAP access) or other LDAP directory can be used for the user registry. The LDAP directory product must be supported by WebSphere Application Server. This includes both Domino and all IBM SecureWay LDAP directory servers.
- ▶ All users must be defined in a single LDAP directory. Connecting more than one directory together using LDAP referrals is not supported. Using multiple Domino Directory Assistance documents to access multiple directories is also not supported.
- ▶ The users' Web browsers must have HTTP cookies enabled since the authentication information that is generated by the server is transported to the Web browser in a cookie. The cookie is then used to propagate the user's authentication information to other servers, relieving the user from entering the authentication information for every request to a different server.
- ▶ Domino R5.0.6a for AS/400 (or later) and Domino R5.0.5 (or later) for other platforms are supported.
- ▶ A Notes client R5.0.5 (or later) is required for configuration of the Domino server for SSO.
- ▶ Authentication can be shared across multiple Domino domains.
- ▶ WebSphere Application Server V3.5.1 (or later) for all platforms is supported.
- ▶ Any HTTP Web server supported by WebSphere Application Server.
- ▶ Authentication can be shared across multiple WebSphere administrative domains.
- ▶ Both the WebSphere Application Server Standard and Advanced Editions are supported. Since configuration is the same for both editions, a distinction between the editions will not be made in the information provided here.
- ▶ Basic authentication (user ID and password) using either Basic or Custom Challenge Types is supported.
- ▶ Permissions for either all authenticated users or groups of users is supported. If you are using the Domino Directory for authentication, and have not specified a Base Distinguished Name during setup permissions for individual users is also supported.

**Note:** At this time, when using other LDAP directories, SSO only supports selection by group, which means that the Web user must be a member of a group that is defined in the LDAP directory.



To enable SSO between Domino servers and WebSphere Application Servers, you must first configure SSO for WebSphere and then configure SSO for Domino. The necessary configuration steps are covered in the following sections.

## 5.3 Which LDAP server should be used?

Lotus Domino and the WebSphere Application Server do not necessarily need LDAP to authenticate users. However, if you want to enable single sign-on between Domino and WebSphere, you need to have a single user registry, which can be accessed by both servers. This can only be accomplished by using a user directory which can be accessed by LDAP.

In an iSeries server environment we have the choice between using the SecureWay Directory for OS/400, which the LDAP Server implemented in OS/400 or the LDAP server capabilities of Lotus Domino for AS/400.

Refer to Chapter 4, “Directory sharing using LDAP” on page 57 for information on how to enable and use either the OS/400 LDAP server or the Domino LDAP support for Domino and WebSphere.

## 5.4 Configuring SSO for WebSphere

Enabling single sign-on is only a very small part of the whole process to activate WebSphere authentication. In Chapter 4, “Directory sharing using LDAP” on page 57 we stepped through how to enable security for the WebSphere Application Server using either the OS/400 LDAP server or the Domino LDAP support.

To prepare to use the single sign-on abilities for WebSphere and Lotus Domino for iSeries, you must update the WebSphere global security configuration again with single sign-on enabled, and re-generate and export the LPTA keys to be used when you configure Lotus Domino for iSeries for SSO.

### 5.4.1 Updating the global security settings for WebSphere

To use SSO with Domino and WebSphere Application Servers, you must first configure SSO for WebSphere. SSO for WebSphere allows authentication information to be shared across multiple WebSphere administrative domains and with Domino servers.

If you would like to provide SSO to WebSphere Application Servers in more than one WebSphere administrative domain, you'll need to configure each of the administrative domains to use the same DNS domain, user registry (using LDAP) and a common set of LTPA keys as described below.

**Note:** This section assumes that you have already installed the WebSphere Application Server, configured one or more application servers in one or more WebSphere administrative domains and have enabled global security (without SSO enabled) as outlined in Section 4.3.2, “Enabling global security in the WebSphere Application Server” on page 81, for OS/400 LDAP or in Section 4.5.2, “Enabling global security in the WebSphere Application Server” on page 103, for Domino LDAP.

To enable SSO in WebSphere, perform the following steps:

1. On the WebSphere Administrative Console, click the **Wizard** icon and select **Configure Global Security Settings** (Figure 5-1).

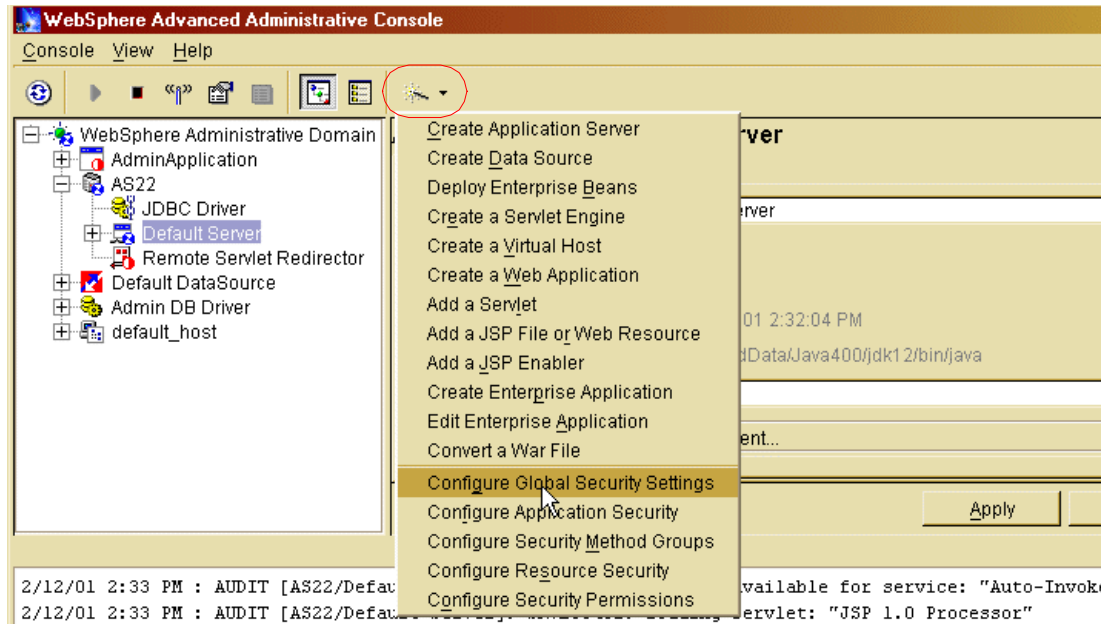


Figure 5-1 Configure Global Security Settings

Because you previously configured WebSphere global security settings, there are only a few things that need to be changed.

2. Click the **Authentication Mechanism** tab (Figure 5-2).

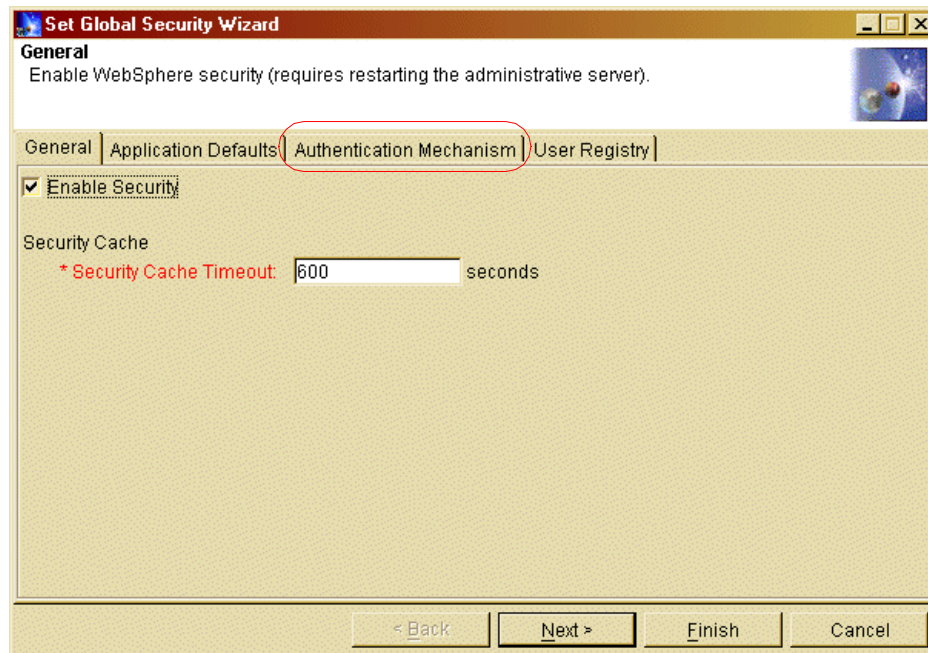


Figure 5-2 Global Security Settings: General tab

3. On the Authentication Mechanism window (Figure 5-4):
  - a. Ensure that Lightweight Third Party Authentication (LTPA) is selected.
  - b. Select the **Enable Single Sign On (SSO)** box to enable SSO and authentication information to be placed in HTTP cookies.

- c. Set Domain to the domain portion of your fully qualified Internet name. In our example, this is `itsoroch.ibm.com`.

**Note:** WebSphere Application Server treats the DNS domain as case sensitive, so ensure that the DNS domain value is specified exactly the same, including casing, whenever you use the value.

- d. Re-generate the LTPA keys to be used by the WebSphere administrative domain that you are configuring. Click the **Generate Keys** button to generate keys for LTPA.
- e. When prompted, enter the LTPA password. This password is associated with the LTPA keys. Click **OK** to save the LTPA keys (Figure 5-3).

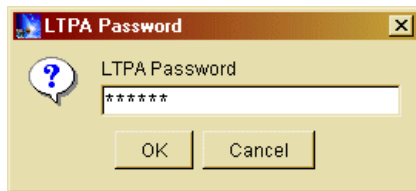


Figure 5-3 LTPA keys password

Remember this password because it is used later when importing these keys while configuring SSO for Domino.

4. Make sure you see the message Command “Generate LTPA Keys” completed successfully in the Console Messages window, then click **Finish** to save the updated global security settings.

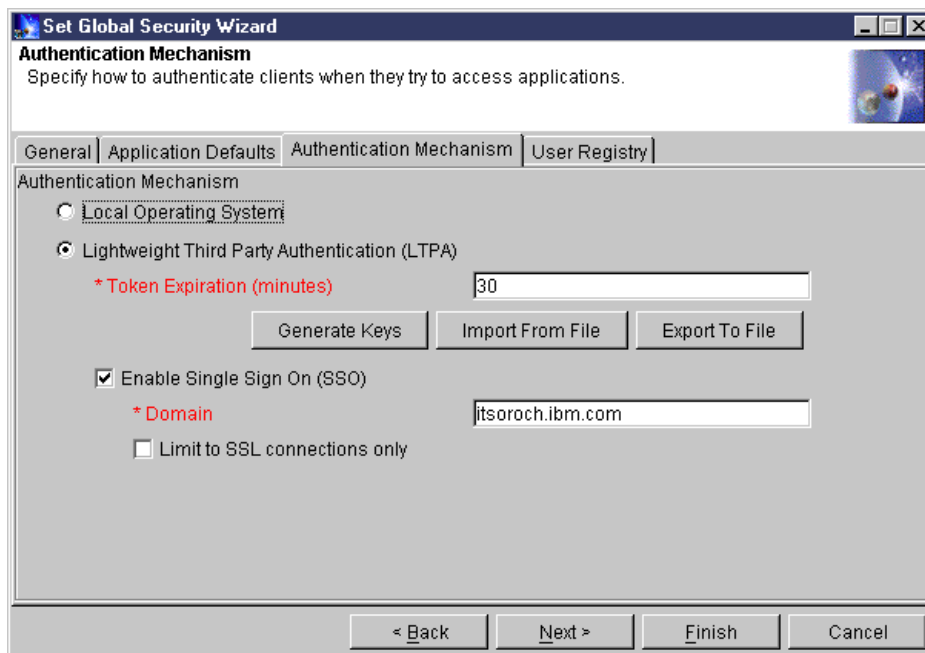


Figure 5-4 Global Security setting: Authentication Mechanism for SSO

5. Click **OK** on the information message window that warns about changes not taking effect until the administrative server is restarted (see Figure 5-5).

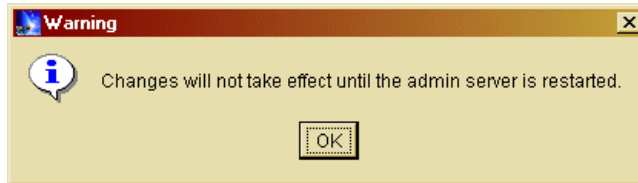


Figure 5-5 Changing security: Warning window

6. Right-click your node, and select **Restart** (see Figure 5-6).

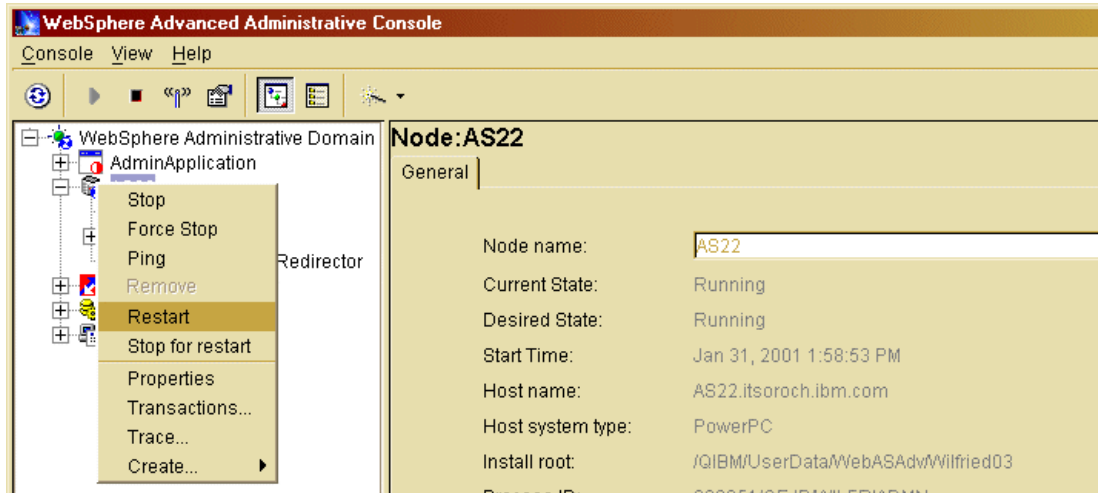


Figure 5-6 Restarting the administrative server

7. Click **Yes** on the confirmation window (see Figure 5-7).

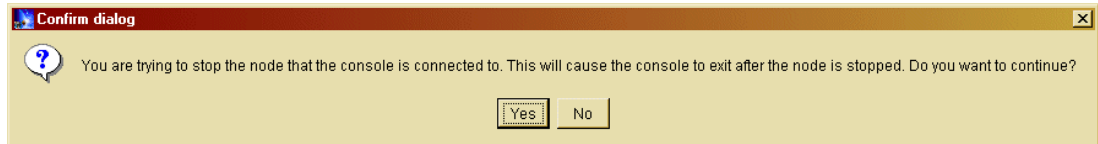


Figure 5-7 Restart warning window

8. Monitor the WASxxADMN administrative server task (or job) from the Work with Active Jobs window (WRKACTJOB) to ensure that the WebSphere Administrative server restarts successfully. As you watch the Administrative server job, notice that it stops, starts, stops, and then starts again. This is expected after global security settings have been changed. The entire process may take several minutes or more.
9. Once the WebSphere Administrative Server has successfully and completely restarted, start the WebSphere Administrative Console. Specify the user ID and password exactly as you configured previously for the Security Server ID and Security Server Password fields in the global security settings (Figure 5-8).

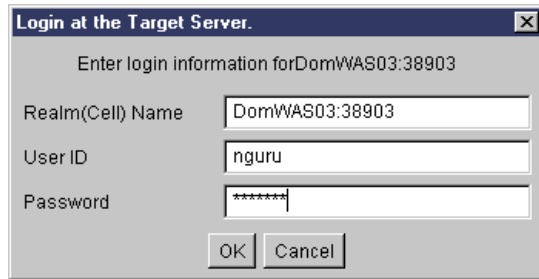


Figure 5-8 Sign on to the WebSphere Administrative Console

## 5.4.2 Exporting the LTPA keys

To complete the WebSphere security configuration for SSO, export the LTPA keys to a file. This file is used later when importing keys into the Domino Web SSO configuration document. Perform the following steps:

1. On the WebSphere Administrative Console, click the **Wizard** icon and select **Configure Global Security Settings**.
2. Click the **Authentication Mechanism** tab.
3. Click the **Export To File** button to export the LTPA keys to a file.
4. On the Export to File window, specify a LTPA Keys file name and a location to contain the LTPA keys. Any file name and extension works. However, be sure you remember it since you use it later (see Figure 5-9).

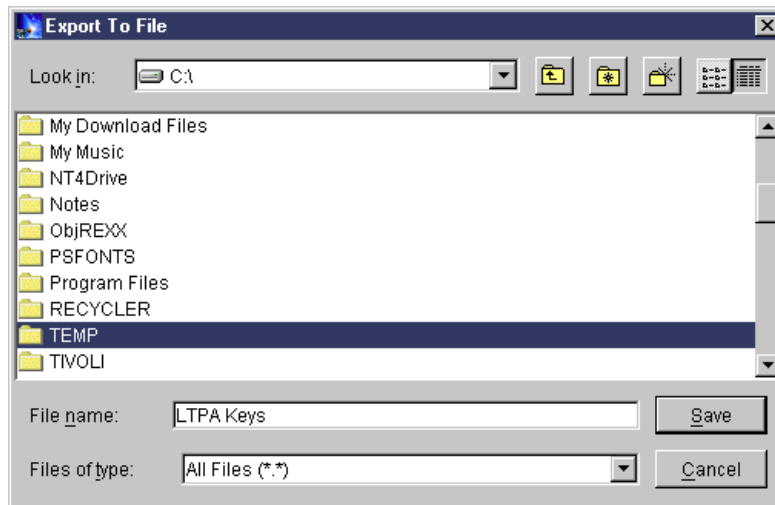


Figure 5-9 Exporting LTPA keys to a file

5. Click **Save** to save the file.
6. Click **Cancel** to close the Global Security Settings wizard.

## 5.5 Configuring SSO for Domino

Configuring SSO for Domino is accomplished by selecting a new Multi-server option in the Domino server document for session-based authentication, along with creating a new domain-wide configuration document in the Domino Directory called the Web SSO Configuration document. The Web SSO Configuration document, which should be replicated to all Domino servers participating in the SSO domain, is encrypted for participating Domino servers and contains a shared secret used by Domino servers for authenticating user credentials.

**Note:** Before you can configure Domino for single sign-on with WebSphere, you need to configure SSO for WebSphere first (see Section 5.4.1, “Updating the global security settings for WebSphere” on page 125), because the LTPA keys generated by WebSphere (refer to Section 5.4.2, “Exporting the LTPA keys” on page 129) have to be imported into Domino.

### 5.5.1 Creating the Domino Web SSO Configuration document

To create the Domino Web SSO Configuration document, use a Lotus Notes client R5.0.5 (or later) and perform the following steps:

**Note:** When you perform the following steps with a Notes client, you need to make sure the location document of your Notes client points to the Domino server where you want to enable SSO. This is needed so that public key can be used for the server. If a message appears when you save the Web SSO Configuration document saying it could not find server, then this should fix the message. Otherwise, if you try starting the HTTP server after enabling Multi-Server in Session authentication then you will get the Error Loading Web SSO configuration message when HTTP is started.

You may also have to add the Domino server's names.nsf to the existing NAMES= line in notes.ini file of your Lotus Notes client as follows:

```
NAMES=names.nsf,CN=DomWASxx/O=Domxx!names.nsf
```

1. From the Lotus Notes client, open the **Domino Directory** database (names.nsf) of your server Domino server and select the **Servers** view.
2. Click the **Web...** action button and select the **Create Web SSO Configuration** to create the document (Figure 5-10).

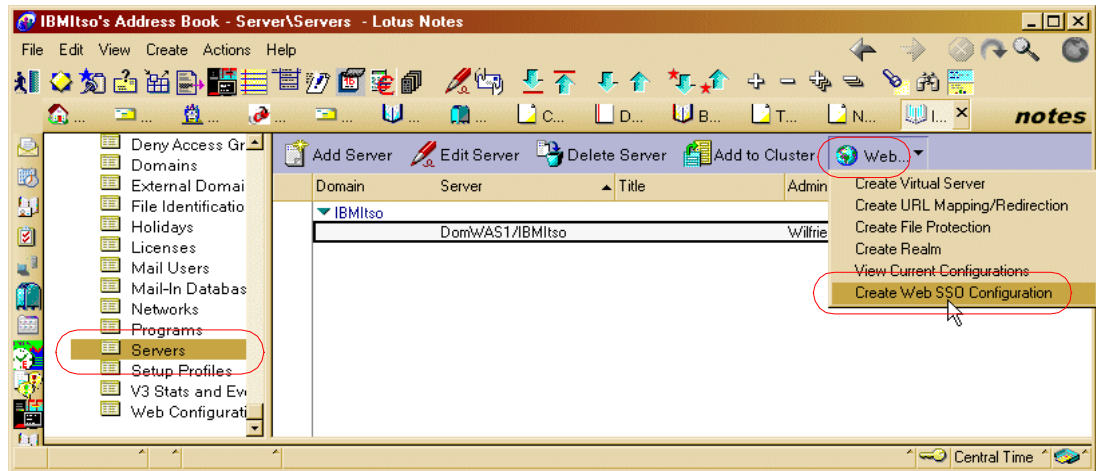


Figure 5-10 Selecting Create Web SSO Configuration

3. In the Web SSO Configuration document, click the **Keys...** action button as shown in Figure 5-11.

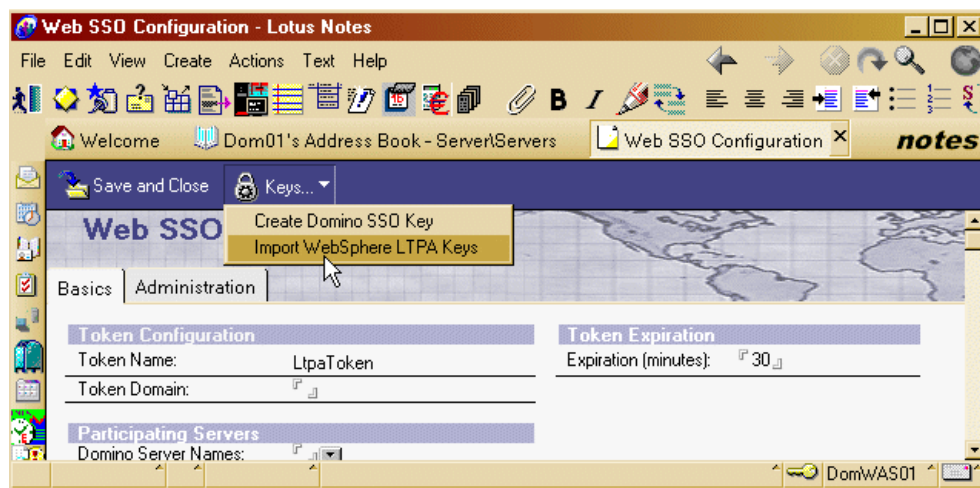


Figure 5-11 Selecting Keys -> Import LPTA Keys on the Web SSO document

4. Select **Import WebSphere LTPA Keys** to import the LTPA keys from a file.
5. Enter the path to the LTPA Keys file that was exported earlier (Figure 5-12) and click **OK** to import the file. Refer to Section 5.4.2, "Exporting the LTPA keys" on page 129, for details.

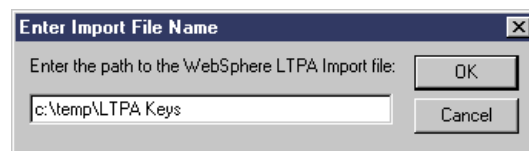


Figure 5-12 Entering the import file name for LTPA keys

6. Enter the password that you used earlier when generating the LTPA keys (Figure 5-13).

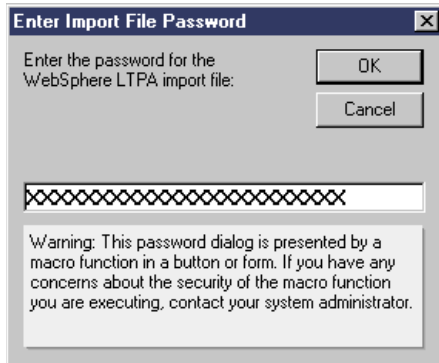


Figure 5-13 Enter LTPA keys password

7. Click **OK** on the Successfully imported WebSphere LTPA keys message (Figure 5-14).

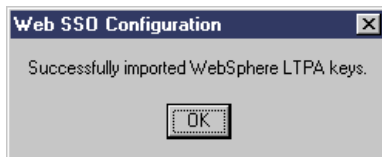


Figure 5-14 Imported WebSphere LTPA keys successful message

8. The Web SSO Configuration document should automatically be updated to reflect the information from the LTPA Keys file you just imported.
9. Figure 5-15 shows the Web SSO Configuration document. Update the fields in this document as follows:

- **Token Domain**

This is the DNS domain portion of the fully qualified Internet name of your system. Because all servers participating in SSO must be in the same DNS domain, this value must be the same as the Domain value specified when configuring WebSphere Application Server.

**Note:** WebSphere Application Server treats the DNS domain as case sensitive, so ensure that the DNS domain value is specified exactly the same, including casing, whenever you use the value.

- **Domino Server Names**

These are the Domino servers that will be participating in SSO. This document will be encrypted for the creator of the document, the members of the Owners and Administrators fields, and the Domino servers specified in this field.

**Note:** You must specify a fully qualified Domino server name here (for example, MyDominoServer/MyOu). The Domino server name that you specify here must also match the name of the Home/mail server currently in the active Location document on your Lotus Notes client. Some documentation mentions the Connection document at this point, which is incorrect.

- **LDAP Realm**

The fully qualified host name of the LDAP server. This field is initialized from the information provided in the LTPA keys file.



**Note:** You only need to change this value if an LDAP server port value was specified for the WebSphere administrative domain. If a port is specified, a backslash (\) must be inserted in the value before the colon. For example, replace `mymachine.mydomain.ibm.com:389` with `mymachine.mydomain.ibm.com\ :389`.

## – Token Expiration

This is the number of minutes a token can exist before expiring. A token does not expire based on inactivity. Rather, it is valid for only the number of minutes specified from the time of issue.

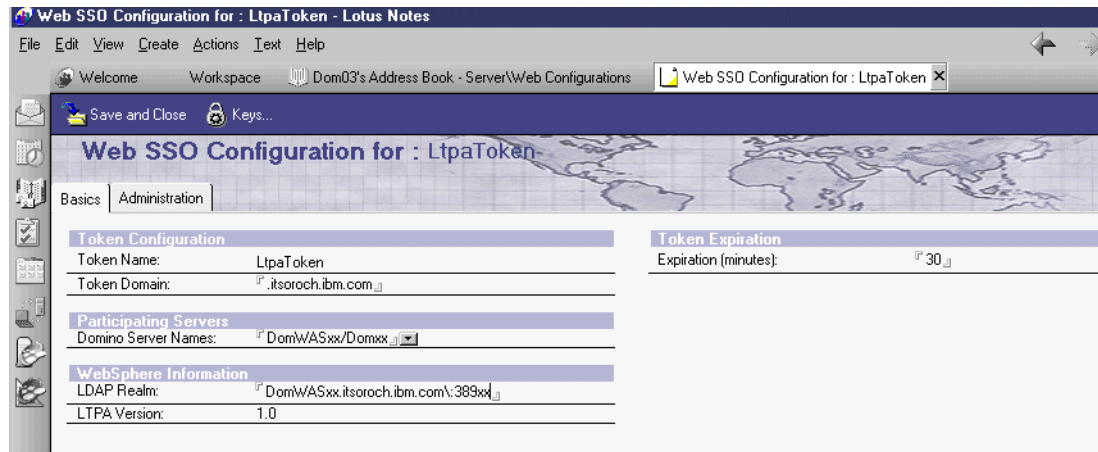


Figure 5-15 Domino Web SSO Configuration document

10. Click **Save and close** to close the Web SSO Configuration document. It now appears in the Web Configurations view.

## 5.5.2 Configuring the Domino server document

To update the Domino server document for SSO, perform the following steps:

1. In the Domino Directory, edit the Domino server document. See Figure 5-16.

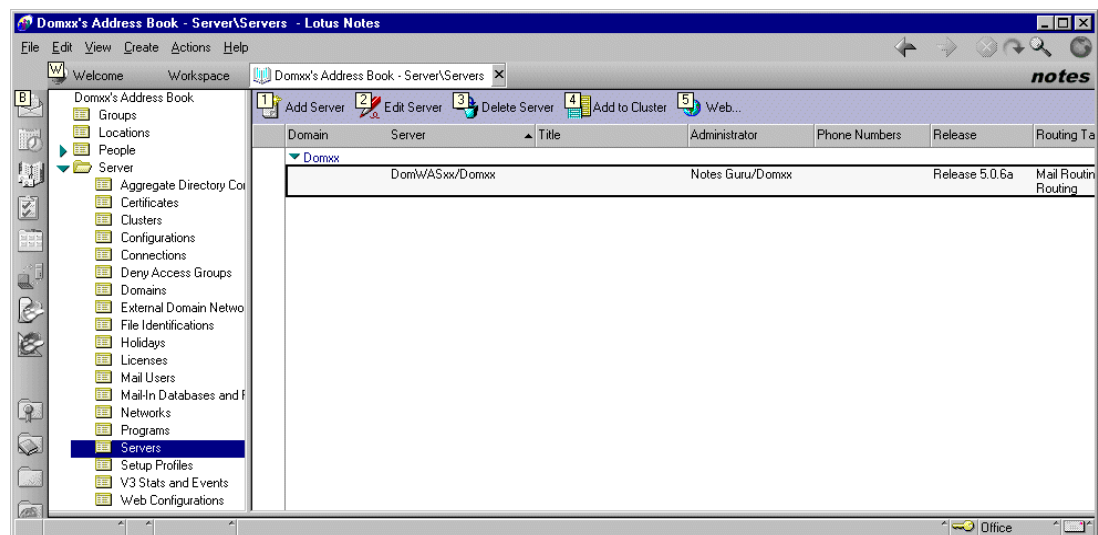


Figure 5-16 Domino server view

2. Select the **Ports** tab, then the **Internet Ports** sub-tab, and then select the **Web** sub-sub-tab. See Figure 5-17.

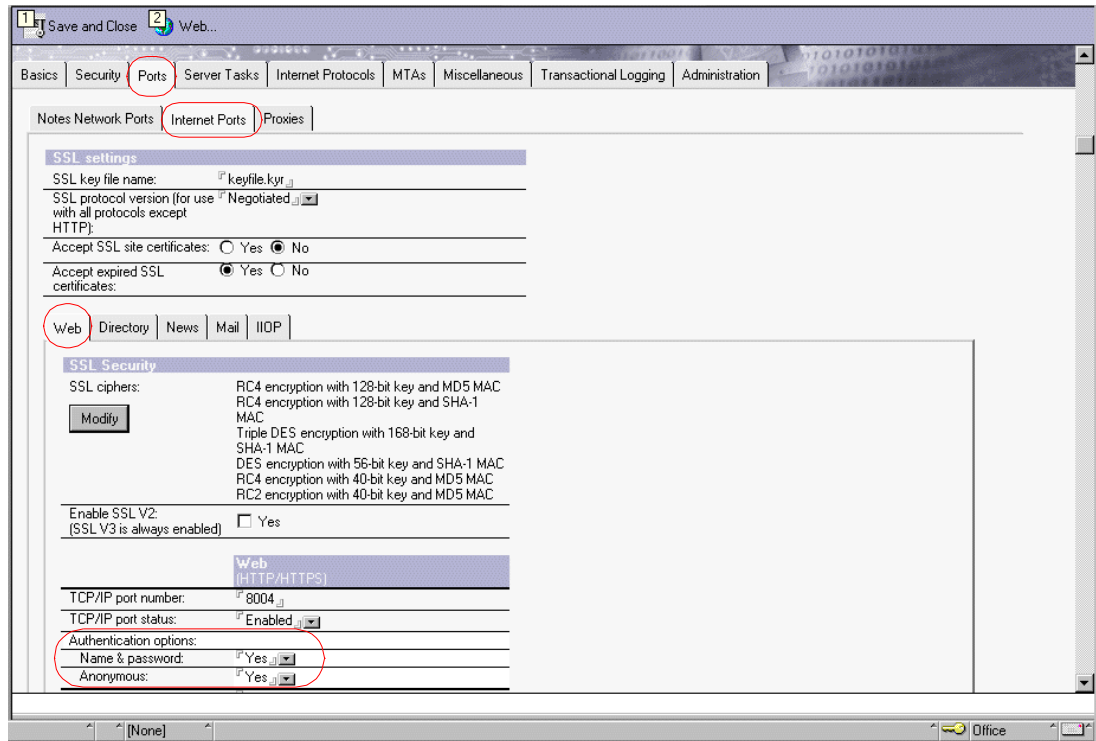


Figure 5-17 Ports, Internet Ports, and Web tabs selected

3. Verify the Name & password field in the TCP/IP Authentication options section is set to **Yes**. This enables basic authentication for Web users. See Figure 5-17.
4. Select the **Internet Protocols** tab, and then select the **Domino Web Engine** sub-tab (Figure 5-18).

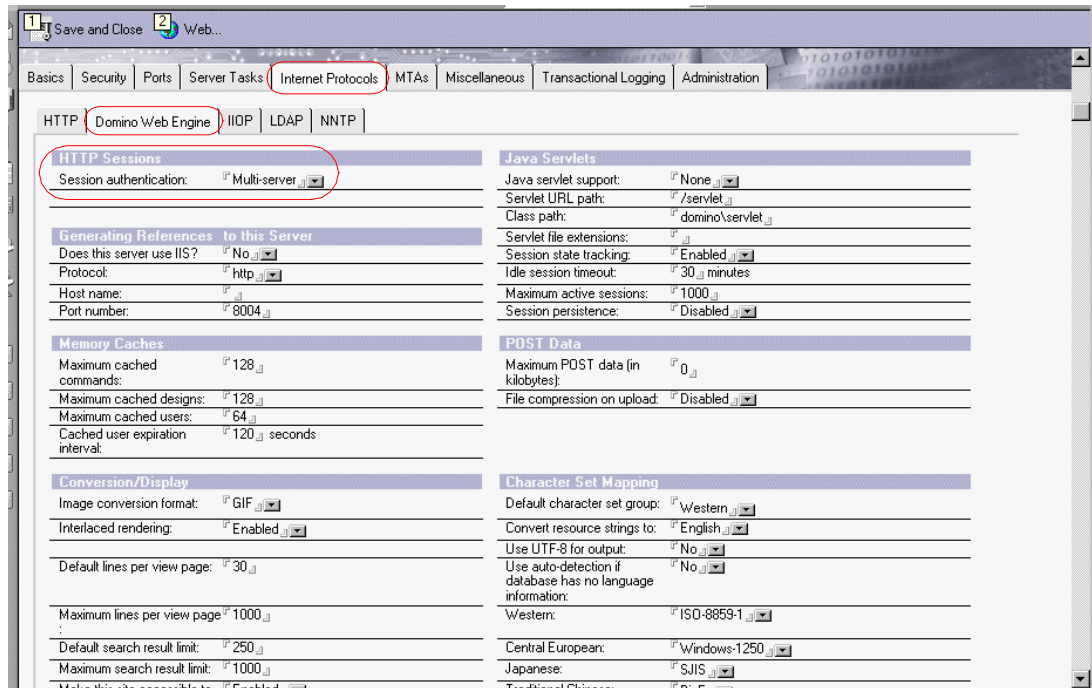


Figure 5-18 Selecting Internet Protocols and the Domino Web Engine

5. In the HTTP Sessions section, click the drop-down menu in the Session authentication field and select **Multi-server**. This enables SSO for Domino (Figure 5-18).
6. Click **Save and Close** to close the Domino server document.
7. For these changes to take effect, the Domino server must be stopped and restarted. To do this, from your 5250 session, enter the following command and press Enter:  
 WRKDOMSVR DomWASxx  
 where DomWASxx is your Domino server name.
8. From the Work with Domino Servers window, enter option 6 (End Server) next to your Domino server.
9. Refresh the window to verify that the Domino server is ended by pressing F5 until the status is \*ENDED.
10. Enter option 1 (Start Server) next to your Domino server to start it again.

**Note:** If you are running the Domino HTTP sever, You will see the following message when the HTTP server task starts:

HTTP: Successfully loaded Web SSO configuration

If a Domino server enabled for SSO cannot find a Web SSO configuration document or is not included in the Domino Server Names field (and thus cannot decrypt the document), then the following message should appear on your Domino server's console:

HTTP: Error Loading Web SSO configuration. Reverting to single-server session authentication.

## 5.6 Verifying single sign-on between WebSphere and Domino

At this point you are ready to verify that SSO for WebSphere and Domino is configured and working correctly. In this section we provide examples of using SSO between Domino and WebSphere Application Servers.

Before proceeding with your verification of SSO between WebSphere and Domino, you should have already verified the following items:

- ▶ The LDAP directory contains at least one user defined for testing purposes.
- ▶ The WebSphere Administrative Console can be started for each the WebSphere administrative domains that you are using. You should be authenticated to the administrative domain using the security name associated with a user defined in the LDAP directory.
- ▶ At least one user in the LDAP directory must be authorized to access at least one Domino resource, such as the Domino Directory.
- ▶ At least one user in the LDAP directory must be authorized to access at least one WebSphere application server resource, such as the hello servlet.
- ▶ From a Web browser that is configured to not accept HTTP cookies, you should be able to access resources, such as servlets, that are protected by each of the WebSphere Application Servers. You should be prompted for a user ID and password.
- ▶ You should be able to access a resource, such as a Domino database, that is protected by each of the Domino servers. You should be prompted for a user ID and password.

After performing the verifications listed above, you are now ready to verify that SSO is working correctly as follows:

1. Configure the Web browser to accept HTTP cookies. If you are using Internet Explorer, the type of cookies that need to be enabled are per-session (not stored).
2. For testing purposes only, configure the Web browser to prompt before accepting HTTP cookies. This will provide you with visual confirmation that the Domino and WebSphere application are generating and returning HTTP cookies to your Web browser after you authenticate.
3. From the Web browser, specify the URL for a resource protected by the Domino server. For example, open a database which defines no access for user anonymous. This will verify that the token generated by the Domino server is accepted by WebSphere application servers. Make sure to enter the fully qualified TCP/IP host name for the URL. For example, enter `http://myhost.mycompany.com/names.nsf` and not just `http://myhost/names.nsf`.

When prompted for a user ID and password, make sure that you specify a user ID that is authorized to resources for both the Domino and WebSphere application servers. The format of the name that you specify depends on the level of restriction Domino is using for Web users and whether a Domino Directory or another LDAP directory is being used.

Refer to the *Controlling the level of authentication for Web clients* document in Domino 5 Administrative Help for details on the options for specifying a user name for basic authentication. The level of restriction Domino uses for Web users is set in the Web server authentication field found on the Security tab of the Server document. If you are using the default configuration settings, you should specify the user's short name or user ID. Accept the HTTP cookie when prompted. You should now have access to the resource.

4. From the same Web browser session, access a resource protected by a WebSphere Application Server. You should have access to the resource without being prompted for a user ID and password. If you are prompted, refer to SSO fails when accessing protected

resources for assistance. Make sure to enter the fully qualified TCP/IP host name for the URL. For example, enter `http://myhost.mycompany.com/webapp/examples/showCfg` and not `http://myhost/webapp/examples/showCfg`.

5. From the same Web browser session, access resources managed by the remaining Domino and WebSphere application servers in your domain configured for SSO.
6. Restart your Web browser session and perform the SSO verifications steps mentioned above, but this time access a resource protected by a WebSphere Application Server first. This will verify that the token generated by WebSphere application server is accepted by the Domino server(s). When prompted for a user ID and password, use the user's shortname or user ID since this is the default WebSphere Application Server naming convention for users.

In the following sections we provide some examples of a WebSphere servlet linked to a Domino application. We begin by first showing you the two applications enabled with security, but *without* the single sign-on function enabled.

We then show you the two applications *with* the single sign-on function enabled. The first example shows accessing the servlet in WebSphere and then linking to the Domino application. With SSO you are only prompted to sign on once when initially going to the WebSphere servlet. After that, you are able to move back and forth between the two applications without being prompted to sign on again.

The second SSO example shows accessing the Domino application first and then linking to the servlet in WebSphere. Again, you are only prompted once when initially going into the Domino application. After that, you are able to move back and forth between the two applications without being prompted to sign on again.

### 5.6.1 Verifying Domino and WebSphere security without SSO enabled

This section provides an example of a Web browser accessing a WebSphere servlet that is linked to a Domino application. The two applications are enabled with security, but *without* the single sign-on function enabled.

1. We begin by opening a Netscape browser and entering the URL to access the WebSphere servlet, for example:

`http://AS22.itsoroch.ibm.com:80xx/webapp/DomApp/SimpleServlet`

2. We are prompted to sign on in order to access the WebSphere servlet, as shown in Figure 5-19.

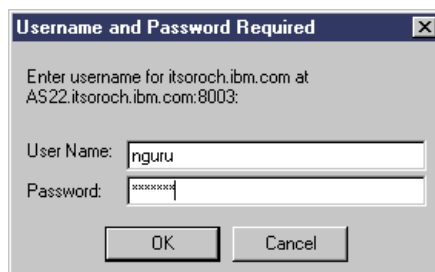


Figure 5-19 WebSphere servlet sign on prompt

3. In this example we are prompted to accept a cookie (see Figure 5-20).

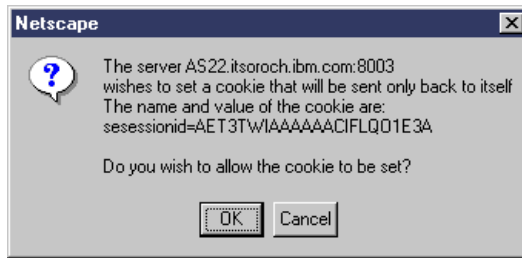


Figure 5-20 Cookie

4. The WebSphere servlet Web page now appears (see Figure 5-21).

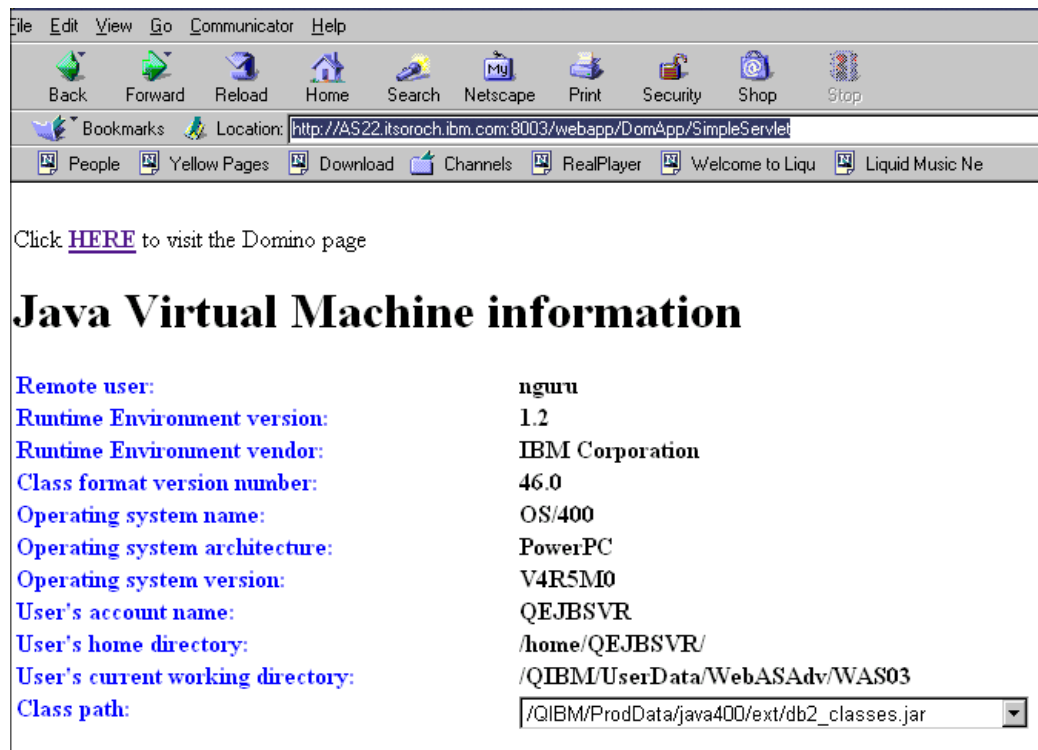


Figure 5-21 WebSphere servlet

5. To access the Domino application from the WebSphere servlet in our example, we click the link:  
Click [HERE](#) to visit the Domino page
6. Since we have not yet enabled the single sign-on capability between WebSphere and Domino yet, we are prompted by Domino to sign on again to access the Domino application (see Figure 5-22).

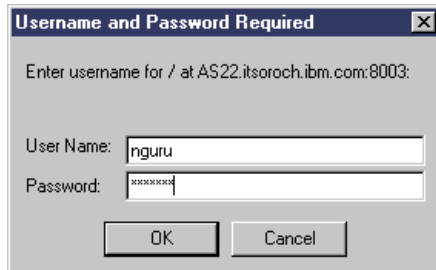


Figure 5-22 Sign on to the Domino application

7. The Domino application window appears (see Figure 5-23).

Figure 5-23 Domino application

8. We can now go back and forth between the Domino application and the WebSphere servlet without having to sign on to either environment again.

## 5.6.2 Verifying Domino and WebSphere security with SSO enabled

This section provides two examples of a Web browser accessing a WebSphere servlet that is linked to a Domino application and vice versa. The two applications are enabled with security *with* the single sign-on function enable. The first example shows accessing the servlet in WebSphere and then linking to the Domino application. With SSO you are only prompted to sign on once when initially going to the WebSphere servlet. After that, you are able to move back and forth between the two applications without being prompted to sign on again.

The second SSO example shows accessing the Domino application first and then linking to the servlet in WebSphere. Again, you are only prompted once when initially going into the Domino application. After that, you are able to move back and forth between the two applications without being prompted to sign on again.

## WebSphere to Domino SSO example

1. We begin by opening a Netscape browser and entering the URL to access the WebSphere servlet. For example:

`http://AS22.itsoroch.ibm.com:80xx/webapp/DomApp/SimpleServlet`

2. We are prompted to sign on in order to access the WebSphere servlet, as shown in Figure 5-24.

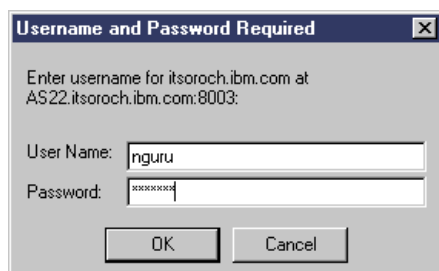


Figure 5-24 WebSphere servlet sign on prompt

3. We are now prompted to accept a cookie. This cookie is generated as a result of SSO. Click **OK**.

**Note:** if you do not see the SSO cookie as shown in Figure 5-25, but only the one as shown in Figure 5-26, your single sign-on configuration is not working correctly!

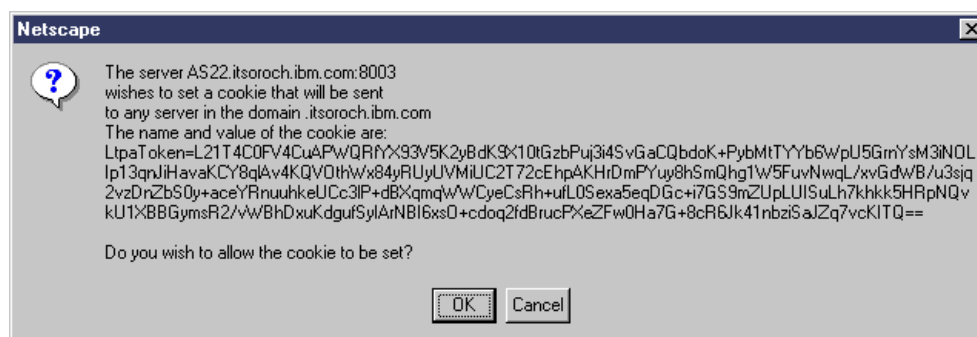


Figure 5-25 SSO cookie

4. We are now prompted to accept another cookie (see Figure 5-26). This is the original cookie that we saw before enabling SSO (refer to the example in Section 5.6.1, “Verifying Domino and WebSphere security without SSO enabled” on page 137). Click **OK**.



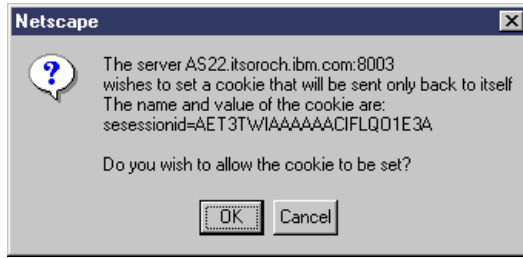


Figure 5-26 WebSphere servlet cookie

5. The WebSphere servlet Web page now appears (see Figure 5-27).

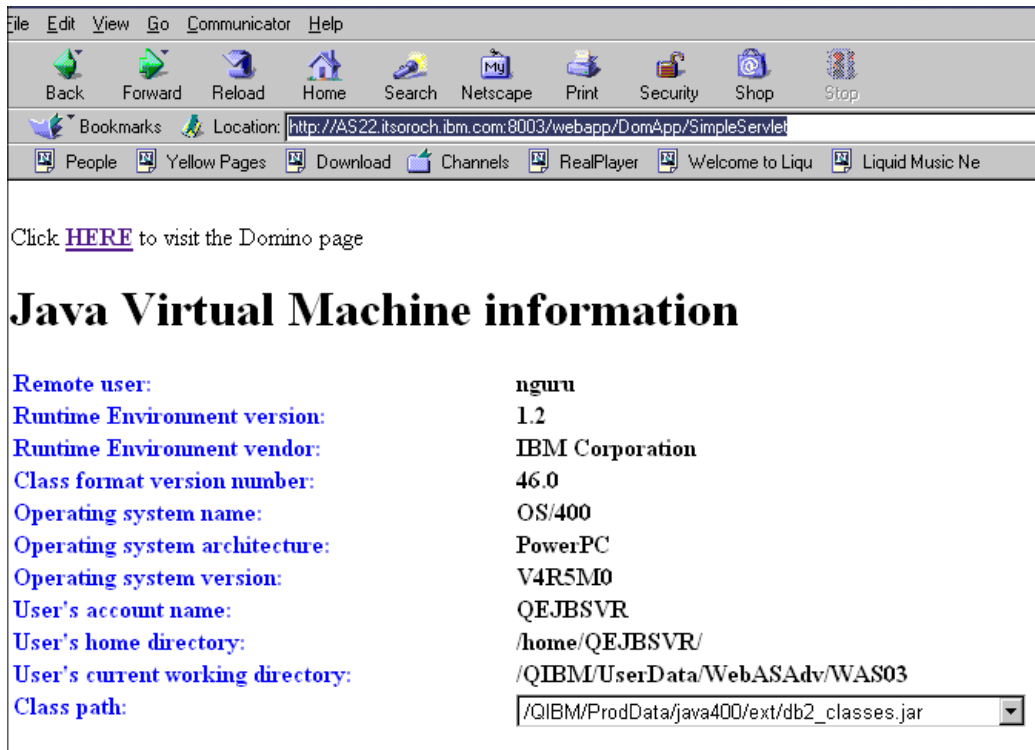


Figure 5-27 WebSphere servlet

6. To access the Domino application from the WebSphere servlet in our example, we click **Click [HERE](#) to visit the Domino page.**

Because we have now enabled the single sign-on capability between WebSphere and Domino, we are not prompted by Domino to sign on again to access the Domino application. Rather, we are taken directly into the Domino application (see Figure 5-28).

Name	Notes Guru	
Account Number	<input type="text"/>	
Address	<input type="text"/>	
City, State, Zip	<input type="text"/>	AK <input type="button" value="v"/>
Loan Tracking Number	DWAS-4UFMPR	
Loan Status	New	
Type of Loan Desired	<input type="radio"/> Personal Loan <input type="radio"/> Car Loan <input type="radio"/> Mortgage	
Term	<input type="radio"/> Please select Loan Type	
Amount Desired	<input type="text"/>	

Figure 5-28 Domino application

7. We can now go back and forth between the Domino application and the WebSphere servlet without having to sign on to either environment again.

### Domino to WebSphere SSO example

1. We now close our Netscape browser window and re-open it to remove the cookies that were set in the previous test.
2. From the Netscape browser, we begin by accessing the Domino Web application by entering its URL, for example:  

```
http://AS22.itsoroch.ibm.com:80xx/DomWASLab.nsf/loanapp?openform
```
3. The Web page shown in Figure 5-29 appears, asking for a valid user name and password.

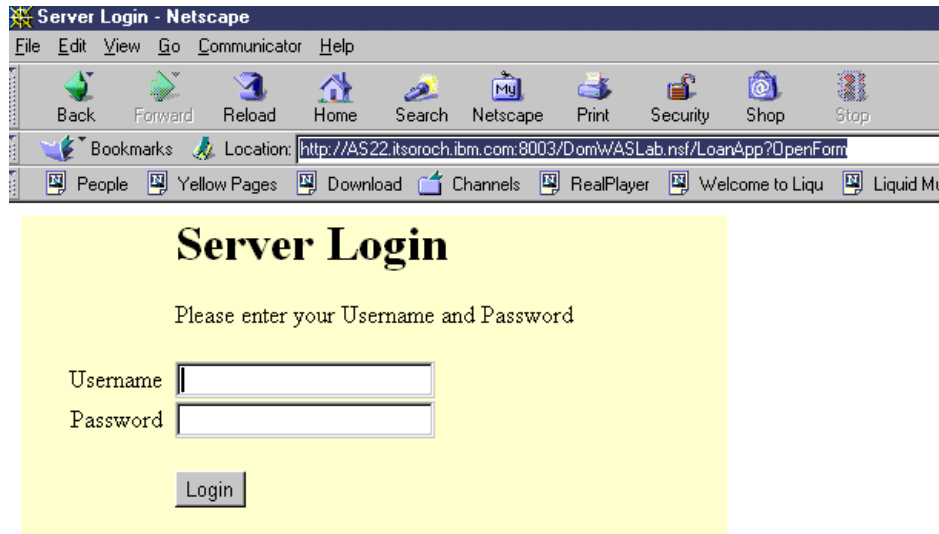


Figure 5-29 Domino SSO challenge

4. We are then prompted to accept a cookie (see Figure 5-30). This cookie is generated as a result of SSO.

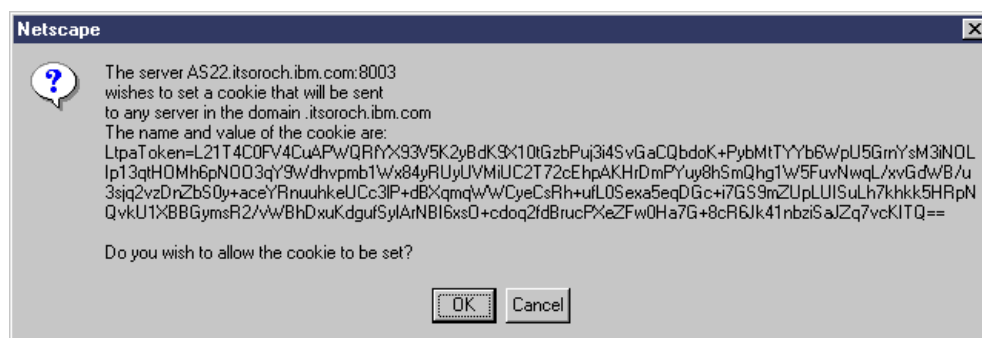


Figure 5-30 SSO cookie

5. The Domino application appears (see Figure 5-31).

**Netscape**  
File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Shop

Bookmarks Location: <http://AS22.itsoroch.ibm.com:8003/DomWASLab.nsf/loanapp?openform>

People Yellow Pages Download Channels RealPlayer Welcome to Liqu

## DWI Bank

Name	Notes Guru	
Account Number	<input type="text"/>	
Address	<input type="text"/>	
City, State, Zip	<input type="text"/>	AK
Loan Tracking Number	DWAS-4UFMPR	
Loan Status	New	
Type of Loan Desired	<input type="radio"/> Personal Loan <input type="radio"/> Car Loan <input type="radio"/> Mortgage	
Term	<input type="radio"/> Please select Loan Type	
Amount Desired	<input type="text"/>	

Figure 5-31 Domino application

6. In our example, clicking the **Submit** button is the link to the WebSphere servlet.
7. On the next Web page that appears (Figure 5-32), we click **Return to the Main Menu**. This takes us to the WebSphere servlet.

**Netscape**  
File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Shop Stop

Bookmarks Location: <http://AS22.itsoroch.ibm.com:8003/DomWASLab.nsf/LoanApp?OpenForm&Seq=1>

People Yellow Pages Download Channels RealPlayer Welcome to Liqu Liquid Music Ne

## Thank you, Notes Guru

[Return to the Main Menu](#)

Figure 5-32 Domino application link to the WebSphere servlet

8. Again, because we have enabled the single sign-on capability between WebSphere and Domino, we are not prompted by WebSphere to sign on again to access the servlet. However, we are prompted to accept another cookie (Figure 5-33). This is the original cookie that we saw before enabling SSO (refer to the example in Section 5.6.1, "Verifying Domino and WebSphere security without SSO enabled" on page 137). Click **OK**.

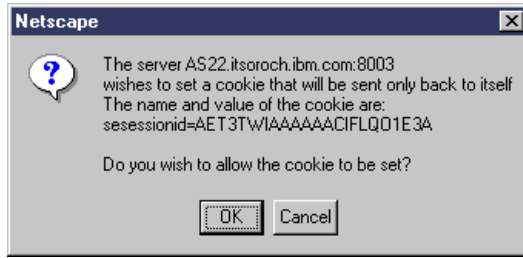


Figure 5-33 Cookie

9. The WebSphere SimpleServlet should appear (see Figure 5-27 on page 141).

## 5.7 Problem determination

This section attempts to provide you with some suggestions for troubleshooting problems in the event you experience problems enabling single sign-on between Domino and WebSphere.

### ***Saving the Domino Web SSO Configuration document fails***

The client must be able to find the Server documents for the participating SSO Domino servers. Since the Web SSO Configuration document is encrypted for the servers you specify, your client's location record's home server must be pointing to a server in the Domino domain where the participating servers reside so that lookups will be able to find the public keys of the servers. If you get a message box that states that one or more of the participating Domino server's cannot be found, then those servers will not be able to decrypt this document and will not perform SSO. When the Web SSO Configuration document is saved, the status bar will state how many public keys were used to encrypt the document by finding the listed servers, authors and administrators on the document.

### ***Domino server fails to load the Web SSO Configuration document***

Since the Domino server document is configured for Multi-Server in the Session Authentication field, the Domino HTTP server will try to find and load a Web SSO Configuration document during startup. The Domino server console will report the following if a valid document is found and decrypted:

HTTP: Successfully loaded Web SSO Configuration.

If participating SSO Domino server(s) are reporting the following, then SSO will not work:

HTTP: Error Loading Web SSO configuration. Reverting to single-server session authentication.

Some possible problems and solutions are:

- Make sure that there is only one Web SSO Configuration document in the Web Configurations view of the Domino Directory and the \$WebSSOConfigs hidden view. Currently you cannot create more than one, but another one could be replicated over from another server. Check the hidden view \$WebSSOConfigs as follows:
  - a. From a Lotus Notes client, select **File -> Database -> Open**.
  - b. In the Open Database window, type in the Domino server name and press Enter, or select the Domino server from the list.
  - c. At the bottom of the Open Database window, in the File Name: field, enter `names.nsf`, but don't press Enter (see the next step).

- d. While holding the shift and Ctrl buttons down, click the **Open** button to open up the Domino Directory with all the hidden views.
- e. At the bottom of the view list, click **\$WebSSOConfigs**.
- f. Make sure you only have one document in this view. If you have more than one, then delete them all and recreate the Web SSO Configuration document.
- If the Server document's public key does not match the public key in the ID file, then the decrypting of the Web SSO Configuration document will fail and you will receive the error message above on the console. This could happen if the ID file was created multiple times and didn't update the Server document correctly. Usually there is an error on the Domino server console that states that the public key does not match the server ID. If this happens then SSO will not work because the document could be encrypted with a public key for which the server does not possess the corresponding private key to decrypt with. The way to correct this is to copy the public key out of the server ID then paste it into the Server document and recreate the Web SSO Configuration document.

### ***Authentication fails accessing a protected resource***

After attempting to authenticate, if the Web user continues to be prompted for a user ID and password, security is not configured correctly. The Domino or WebSphere security server is not able to authenticate the user with the LDAP server. Some possible problems and solutions are:

- Verify that the LDAP server can be accessed from the Domino server. Try using the TCP/IP ping utility to verify TCP/IP connectivity and that the host machine is running.
- Verify that the LDAP user is defined in the LDAP directory. Try using the `ldapsearch` utility to verify that the user exists and that the password is correct. For example, the following command can be run from the OS/400 Qshell, AIX typescript, or Windows DOS prompt:

```
ldapsearch -D "cn=John Doe, ou=Rochester, o=IBM, c=US" -w mypassword -h
ghost.mycompany.com -p 389 -b "ou=Rochester, o=IBM, c=US" (objectclass=*)
```

**Note:** This command has been wrapped for display purposes. Enter it as one line.

A list of directory entries should be displayed by the `ldapsearch` command. Possible errors and causes are:

- “No such object” error means the directory entry referenced by the user's DN value (-D parameter) or the base DN value (-b parameter) does not exist.
- “Invalid credentials” means the password is invalid.
- “Can't contact LDAP server” means the server's host name or port is invalid or the LDAP server is not running.
- An empty list means that the base directory specified by the -b parameter doesn't contain any directory entries.
- If you are using the user's short name (or user ID) instead of the DN, be sure that the directory entry is configured with the short name. For a Domino Directory, this is the Short name/UserID field of the Person document. For other LDAP directories, this would be the `userid` property of the directory entry.
- If Domino authentication fails when using an LDAP directory other than Domino Directory, verify the LDAP server configuration settings in the Directory Assistance document in the Directory Assistance database. Also, verify that the Server document references the correct Directory Assistance document. The following LDAP values specified in the Directory Assistance document must match the values specified for those used in the user registry for the WebSphere administrative domain:

- Domain name
- LDAP host name
- LDAP port
- base DN

Also, the Rules defined in the Directory Assistance document should reference the base DN of the directory containing the directory entries of the users.

You can trace the Domino server's requests to the LDAP server by adding the following line to the server's notes.ini file:

```
webauth_verbose_trace=1
```

After restarting the Domino server, trace messages will be displayed in the Domino server's console as Web users attempt to authenticate with the Domino server.

### ***Authorization fails accessing a protected resource***

After authenticating successfully, if the Web user is shown an authorization error message, security is not configured correctly. Some possible problems and solutions are:

- ▶ For Domino databases, verify the user is defined in the database access control settings. Refer to the Domino Administrative documentation for the correct way to specify the user's DN. For example, for a DN of cn=John Doe, ou=Rochester, o=IBM, c=US, the ACL value would be set to John Doe/Rochester/IBM/US.
- ▶ For WebSphere application server resources, verify the security permissions are set correctly. If granting permissions to selected groups, make sure that the user is defined in the group. For example, you can verify the members of the groups using the following URL to display the directory contents:

```
Ldap://myhost.mycompany.com:389/ou=Rochester, o=IBM, c=US??sub
```

**Note:** This command has been wrapped for display purposes. Enter it as one line.

If you have changed the WebSphere administrative domain's LDAP configuration information (host, port, base DN) since last granting permissions, the existing permissions are probably invalid and need to be recreated.

### ***SSO fails when accessing protected resources***

If the Web user is prompted each time they access a resource, SSO is not configured correctly. The following are some of the possible problems and solutions:

- ▶ WebSphere Application Server and Domino must both be configured to use the same LDAP directory. The HTTP cookie used for SSO stores the full Distinguished Name of the user (DN), for example, cn=John Doe, ou=Rochester, o=IBM, c=US and the DNS domain.
- ▶ If the Domino Directory is being used, Web users must be defined using hierarchical names. For example, update the User name field in the Person document to include John Doe/Rochester/IBM/US as the first value.
- ▶ URLs issued to Domino and WebSphere Application Servers configured for SSO must specify the full DNS server name, not just the host name or an TCP/IP address. For Web browsers to be able to send cookies to a group of servers, the DNS domain must be included in the cookie. The DNS domain in the cookie must match the URL. This is why cookies cannot be used across TCP/IP domains.
- ▶ Domino and WebSphere Application Server must be configured to use the same DNS domain. Verify that the DNS domain value is exactly the same (including casing). The DNS domain value can be found in the Configure Global Security Settings of each WebSphere administrative domain and the Domino Web SSO Configuration document. If

you make a change to the Domino Web SSO Configuration document, replicate the document to all Domino servers participating in SSO.

- ▶ Clustered servers must have the TCP/IP host name populated with the full DNS server name in the Server document for Domino ICM (Internet Cluster Manager) to redirect to cluster members using SSO. If this field is not populated, ICM will redirect URLs to clustered Web servers with only the TCP/IP host name, by default, and will not be able to send the cookie because the DNS domain is not included in the URL. To correct the problem you must edit the Domino server document as follows:
  - a. In the Domino server document, select the **Internet Protocols** tab and select the **HTTP** sub-tab.
  - b. Enter the server's full DNS name in the Host Names field.
- ▶ If an LDAP server port value was specified for WebSphere administrative domain, the Domino Web SSO Configuration document must be edited and a \ must be added to the LDAP Realm field for WebSphere servers. For example, replace:

`myhost.mycompany.com:389`

with:

`myhost.mycompany.com\:389`





## Part 2

# Programming topics

In this part of the redbook we focus on the application development programming issues of integrating Domino and WebSphere. This part includes two chapters:

- ▶ Chapter 6, “Using WebSphere and the Domino object models” on page 151
- ▶ Chapter 7, “Accessing Domino from WebSphere” on page 183





## Using WebSphere and the Domino object models

In this chapter we discuss how to access iSeries Domino objects from WebSphere applications running on the iSeries server.

**Note:** All the examples used in this chapter were created with VisualAge for Java and are available for download from the IBM Redbooks Web site. See Appendix B, “Additional material” on page 295, for more information.

## 6.1 WebSphere topology

This section briefly discusses the topology of a WebSphere Application Server environment as shown in Figure 6-1. For in-depth information on IBM WebSphere and VisualAge for Java environments to create, manage and deploy Web-based applications using methodologies centered around servlets, JavaServer Pages (JSPs) and Enterprise JavaBeans (EJBs) architecture, refer to *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, SG24-5755.

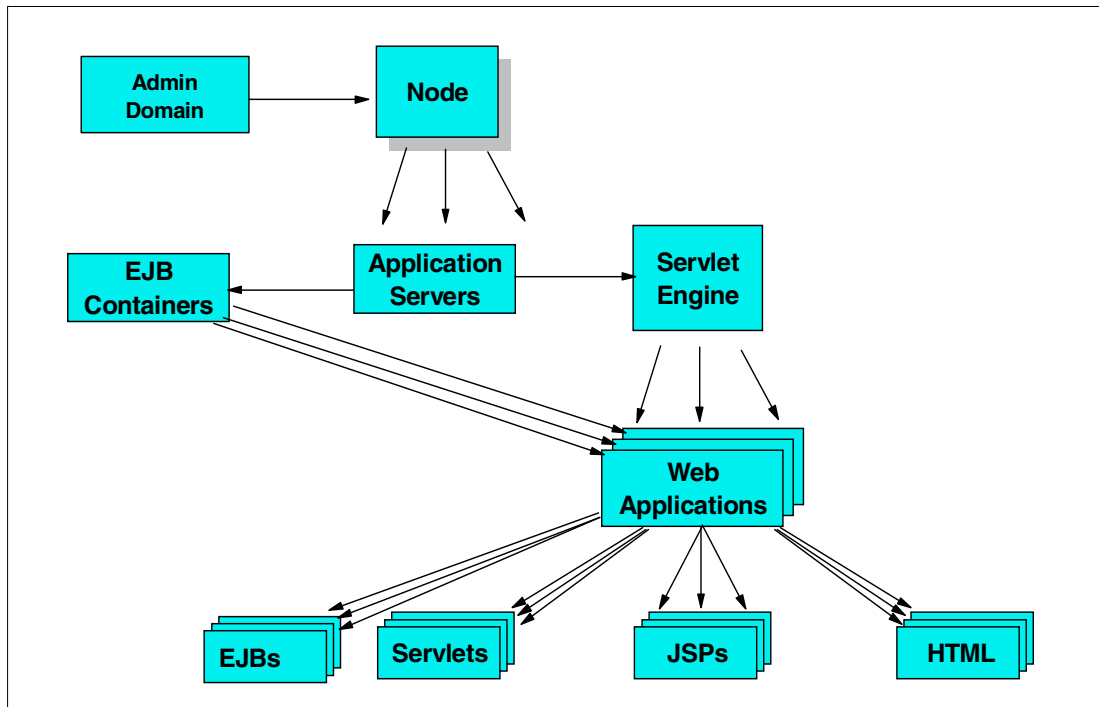


Figure 6-1 WebSphere Application Server topology

The following list provides a brief explanation of the objects shown in Figure 6-1:

- ▶ A node represents a physical machine. After installation, WebSphere Application Server creates a node representing your system, named after the system's host name. For the examples used in this chapter, the node is called "AS22".
- ▶ Application Servers are used to extend the capabilities of a Web server to handle requests for servlets, JSPs, Enterprise JavaBeans (EJBs), and Web applications. It is important to note that WebSphere Application Server is more than just one application server, and can actually be used to support multiple application server processes. An application server in WebSphere Application Server has two main components:
  - A Servlet Engine to handle servlet requests
  - An EJB Container to provide support for EJBs
- ▶ A Servlet Engine is a program that runs within the application server and handles the requests for servlets, JSPs, and other types of server-side include coding. The Servlet Engine is responsible for creating instances of servlets, initializing them, acting as a request dispatcher, and maintaining servlet contexts for use by the Web applications.
- ▶ An EJB container provides an interface between Enterprise JavaBeans and the EJB server, providing many low-level services such as security, threading, support for transactions, and management of data storage and retrieval.

- ▶ A Web application represents a grouping of servlets, JSPs, and their related resources. Managing these elements as a unit allows you to stop and start servlets in a single step. You can also define a separate document root and class path at the Web application level allowing you to keep different Web applications separate in the file system. Servlets that are running within a Web application share the same servlet context with others in the same application, allowing them to communicate with each other.

For brief descriptions of what servlets, JSPs, and EJBs are, refer to “WebSphere programmability” on page 3.

## 6.2 Using Java to access Domino from WebSphere

The WebSphere servlets, JSPs and EJBs that you develop may need to access information or functions that are available in a Domino server environment. The Domino Java API provides two possible separate object models that would allow you to access Domino functionality and services. One choice is to use the Local Domino object model and the other is to use the Remote Domino object model.

This section describes the two Domino Java API object models, how they differ, the factors to consider in deciding which one to use, and the WebSphere modifications needed to enable their usage.

The two Domino API object models share many of the same characteristics. Here are some of the characteristics that they share:

- ▶ Java import statement is the same: `import lotus.domino.*`
- ▶ Class names are the same as well as the method names
- ▶ Each uses a single class file (Notes.jar or NCSOW.jar)

Using the Domino objects requires planning. Depending on the needs of your application, you code the Domino access differently. The questions you must consider are:

- ▶ Will Domino and WebSphere be located on the same machine?
- ▶ Which version of the Domino objects should you use, local or remote?

Which set of objects you use affects the portability as well as the configuration and design of your Java code. While the Java classes are the same, the design is impacted based on which set of objects you use. Both options are available through the identical set of Java interfaces contained in the `lotus.domino` package. However, The two options require using different JAR files (Notes.jar for local access and NCSOW.jar for remote access) and there are differences in the configuration procedure for each. The NCSOW.jar became available with Lotus Domino Release 5.04 and is designed for use with a WebSphere Application Server. Prior to this time, the NCSO.jar was available that allowed remote access to Domino objects but it is not compatible with WebSphere V3 CORBA support. There is also an NCSOC.jar, which is a compressed version of the NCSO.jar that is designed for use in Java applets. It is compressed to allow for faster download of the code by a Web browser.

It is not a trivial exercise to switch from one object model to the other. The following sections describe the differences between setting up WebSphere to use the two Domino object models as well as discussing the trade-offs involved with each option.

**Note:** For additional reference see the article entitled “Tips for Working with Domino Objects” by Bob Balaban in the January 2001 issue of the *WebSphereAdvisor.com* magazine (<http://www.websphereadvisor.com>).

## 6.2.1 Local Domino objects

Local Domino objects assume the code runs on a computer that has a Domino client or server installed on it. It also assumes that the Domino databases reside locally in the file system.

The Java program imports `lotus.domino.*` package and compiles and runs with the `Notes.jar` file. In each operating system platform, there are dependencies on having environment variables set properly and running under the proper operating system user ID or profile. Java code that uses these objects must inherit from the `java.lang.Thread` class or make use of the `NotesThread.sinitThread()` and `NotesThread.stermThread()` functions. This is because Domino requires per-thread initialization and termination. One important aspect of using the `NotesThread` class is that Domino objects are limited to a single thread of execution. This restriction has major design implications and limits the scalability of applications.

Another consideration is that local Domino objects cannot be persisted across servlet invocations because of the single thread restriction. Also when using local objects, you have to be certain to call the `recycle` method prior to terminating the thread to free memory used by the Domino objects. While there are several disadvantages to using the local objects, one advantage is that with local Domino objects you don't have to run the Domino DIIOP service, nor do you have to run Domino's HTTP service.

## 6.2.2 Remote Domino objects

The Common Object Request Broker Architecture (CORBA) is an industry standard developed by the Object Management Group (OMG) that defines a language independent model for exchanging data and discovering services in a networked environment. CORBA can play a significant role in an e-business strategy because it facilitates communication between programs that may be written in various languages. CORBA can enable developers to integrate data from legacy systems with newer programs. For more information on the OMG and to view the CORBA specification, visit: <http://www.omg.org>

Remote Domino objects make use of CORBA to implement access to Domino. In CORBA, communication between objects occurs through Object Request Brokers (ORBs) that use the Internet Inter-ORB Protocol (IIOP) to send messages to each other. In Domino, the Domino Internet Inter-ORB Protocol (DIIOP) service is used for CORBA communication. The advantage of using remote objects is that the Web application server can run on a separate machine from the Domino server. The code still imports `lotus.domino.*`. The Java classes and overall design are essentially the same as local Domino objects. One difference, however, is that the `NotesFactory()` method requires a Domino server IP address/name and a valid user ID and profile that has access to IIOP and the Domino server. The remote Domino server needs to have the DIIOP configured and the user ID must be authorized to use the Domino server and be authorized to run IIOP agents.

Unlike the local Domino objects, the usage of remote Domino objects are not limited to the same execution thread. Domino objects are accessible to all threads.

Remote Domino objects is the preferred method when using the Domino Java API for the following reasons:

- ▶ The servlet can reside on a separate machine from the Domino server.
- ▶ Configuration and execution time requirements are simpler (no `PATH` or user profile requirements).
- ▶ No restrictions on usage in multi-threaded environments.

## Configuring the Domino server for CORBA

Perform the following steps to configure a Domino server for CORBA:

1. To access a Domino server via CORBA, the HTTP and DIIOP server tasks must be running. To load the tasks automatically each time the Domino server is started, edit the notes.ini file and add the tasks to the lists of tasks specified in the ServerTasks parameter, for example:

ServerTasks=<any other tasks>,HTTP,DIIOP

2. Open the Domino server document for editing and choose the **Ports-> Internet Ports -> IIOP** tabs. Figure 6-2 illustrates this section of the Domino server document. Here you can set the port number and enable the port. The default port number is 63149 for a TCP/IP IIOP port and 63148 for the IIOP port that uses SSL. Also, you can specify whether to allow name and password and anonymous access.

The screenshot shows the Domino server document with the 'Internet Ports' tab selected. Within this tab, the 'IIOP' sub-tab is active. The 'SSL settings' section includes fields for 'SSL key file name' (keyfile.kyr), 'SSL protocol version' (Negotiated), and 'Accept SSL site certificates' (No). The 'IIOP Server' section shows 'TCP/IP port number' (63148), 'TCP/IP port status' (Enabled), 'Authentication options' (Name & password: Yes, Anonymous: Yes), 'SSL port number' (63149), 'SSL port status' (Disabled), and 'Authentication options' (Client certificate: N/A, Name & password: No, Anonymous: Yes).

Figure 6-2 Ports-Internet Ports-IIOP tab of Domino server document

3. Configure the number of threads and time-out value. The Time-out Value represents the number of minutes a connection can be idle before being dropped by the server. To set this, choose the **Internet Protocols -> IIOP** tabs of the Domino server document as shown in Figure 6-3.

The screenshot shows the Domino server document with the 'Internet Protocols' tab selected. Within this tab, the 'IIOP' sub-tab is active. The 'Basics' section shows 'Number of threads' (10) and 'Idle session timeout' (60 minutes).

Figure 6-3 Specify IIOP thread count and time-out values

4. Set security for accessing the Domino server and for running Java programs on the server. Figure 6-4 illustrates the Security tab of the Domino server document. In the Server access section, specify who can access the server. If this field is left blank, no one is denied access to the server. If the field has any names listed in it, then only those people or groups specifically listed can access the server. If users are required to log in, this information is checked by the server after the they have been authenticated.
5. In the Java/COM Restrictions section of the Domino server document, specify the users who are allowed to access the Domino objects using CORBA. If this field is left blank, no one is allowed access to Domino via CORBA. The field can accept wildcards. An asterisk (\*) in this field allows everyone. For our example, we specified Test User, a Domino user ID that we created, and Anonymous. Anonymous is a default group that is known internally to Domino as a user who has not been authenticated by the server. Web users are considered anonymous until they log in.

The screenshot shows the 'Security' tab of a Domino server document. The 'Server Access' section is highlighted with a red circle, showing 'Only allow server access to users listed in this Directory' set to 'No'. The 'Java/COM Restrictions' section is also highlighted with a red circle, showing 'Run restricted Java/Javascript/COM' and 'Run unrestricted Java/Javascript/COM' both set to 'Test User, Anonymous'.

Figure 6-4 Security tab of Domino server document

6. If your Domino server is behind a firewall, edit the Domino server's Notes.ini and add the line DIIOP\_IOR\_Host=ipAddress, where ipAddress is the IP address or host name of your Domino server as it is known outside of the firewall.

## 6.3 Configuring a WebSphere Application Server

This section discusses the procedure for configuring the WebSphere Application Server for remote and local access to Domino objects.

### 6.3.1 Configuring access to Domino using remote classes

The following steps show how to configure a WebSphere Application Server Servlet Engine to use remote Domino objects. In this section we configure a sample application. To complete our example you need to have downloaded the following files:

- CustTrack.nsf
- SimpleRemote.java



Refer to the download instructions in Appendix B, “Additional material” on page 295, for information on how to obtain the files. SimpleRemote.java is found in the repository file included with the download materials. Follow the instructions in the appendix for importing the VisualAge for Java repository file.

### Remote example: Before you start

Prior to following the procedure for configuring the WebSphere Application Server, perform the following steps to configure the Domino environment and to modify and compile the servlet using VisualAge for Java:

1. Perform the tasks listed in “Configuring the Domino server for CORBA” on page 155.
2. Place the CustTrack.nsf file into the Domino\Data directory of your Domino server. Change the ownership of the file to the QNOTES user profile by entering the Change Owner (CHGOWN) command on the OS/400 command line. In the example, the command is:  

```
CHGOWN OBJ(' /DOMINO/DOMWAS11/CUSTTRACK.NSF') NEWOWN(QNOTES)
```
3. Create a Person document in your Domino Directory for a user named Test User with an Internet password of domwas. For information on how to create a Person document in Domino, refer to the Domino product documentation.
4. Add Test User to the Java/Com Restrictions section of the Domino server document as described in the “Configuring the Domino server for CORBA” on page 155 step 5. on page 156.
5. Configure a WebSphere Application Server instance. For information on how to do this, refer to the WebSphere Application Server documentation.
6. Edit the SimpleRemote servlet code in VisualAge for Java and modify the value assigned to the serverName variable to match your server name. In our example our server name was domwas11. For information on how to modify code in VisualAge for Java, refer to the VisualAge for Java product documentation.
7. Edit the SimpleRemote servlet code in VisualAge for Java and modify the value assigned to the host variable to match the host name of your server. In our example, our host name was domwas11 and we used port 8011, so the host value was domwas11:8011. If you use the standard HTTP port 80, you do not need to include the port number in the host variable.
8. Export the SimpleRemote class file to a TestDom/servlets directory under the WAS/default\_host directory. The export path is:

```
/QIBM/UserData/WebAsAdv/WASSERVER/default_host/TestDom/servlets
```

Replace WASSERVER with the name of your WebSphere Application Server. This directory was created when you configured your WebSphere Application Server instance. For information on how to export from VisualAge for Java to a directory, refer to the VisualAge for Java product documentation.

For your reference, a full code listing of the SimpleRemote servlet is also available in Appendix A, “Code examples” on page 265.

### Configuring the SimpleRemote servlet

Figure 6-5 illustrates the WebSphere Application Server topology for the remote Domino objects example.

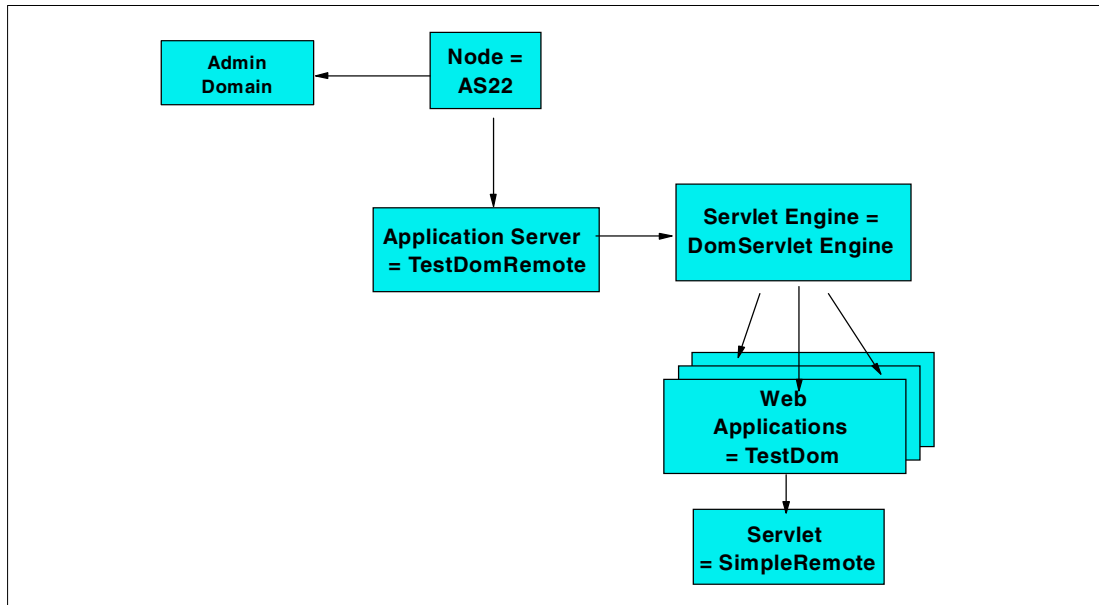


Figure 6-5 WebSphere Application Server topology for Remote Domino access example

1. From your WebSphere Administrative console, create a WebSphere Application Server.  
To do this, right-click the node as shown in Figure 6-6. The example node is called "AS22". Select **Create...** and then select **Application Server**.

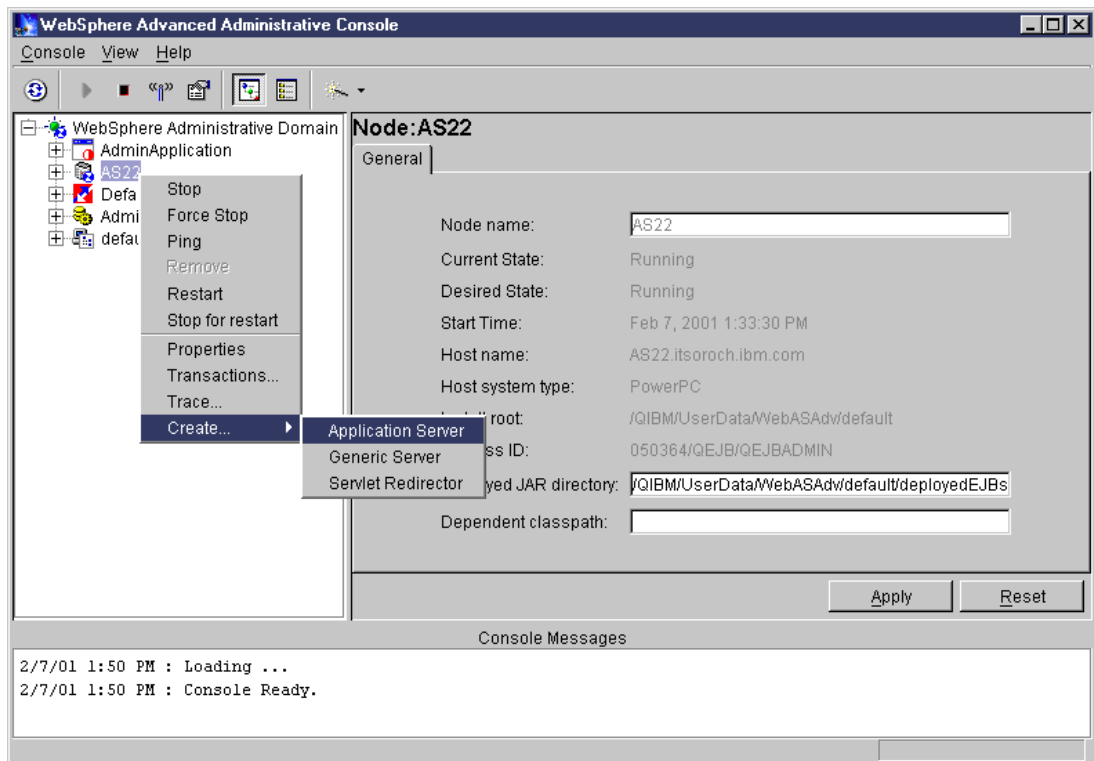


Figure 6-6 WebSphere Advanced Administrative Console

2. Figure 6-7 shows the Create Application Server window that appears. In the Application Server Name field, enter TestDomRemote.

3. In the Command Line Arguments field ,specify the following:

`-Xms32m -classpath /QIBM/ProdData/lotus/notes/data/Domino/Java/NCSOW.jar`

Command-line arguments are passed to the Java virtual machine (JVM) that starts the Application Server process allowing all Web applications access to Domino classes.

Click **OK**.

**Note:** The -Xms parameter sets the initial minimum heap size of the JVM. You can also specify the maximum heap size with the -Xmx parameter. These settings enable you to specify how much memory is available for running your Java applications. The values you set depend on the amount of physical memory on the machine and the size of your Java applications. For more information about tuning the JVM, refer to the “WebSphere Application Server Tuning Guide” in the IBM WebSphere InfoCenter. A starting version of the InfoCenter is installed when you install the WebSphere Application Server. To obtain the full version of the InfoCenter, go to:

<http://www-4.ibm.com/software/webservers/appserv/infocenter.html>

Classpath does not have to be defined here. You could put a classpath in each individual Web application classpath.



Figure 6-7 Create Application Server: General tab

4. Create a Servlet Engine in your WebSphere Application Server. From the WebSphere Administrative Console, right-click your Application Server, select **Create...**, then select **Servlet Engine** as shown in Figure 6-8.

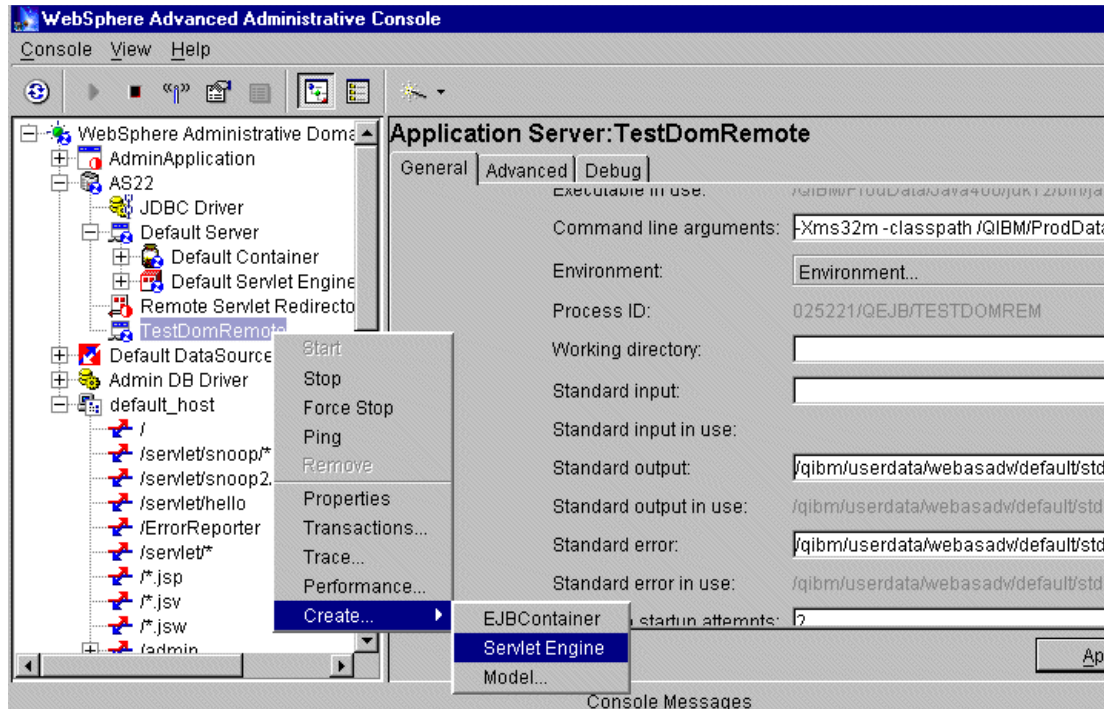


Figure 6-8 Create Servlet Engine in WebSphere Application Server

5. On the Create Servlet Engine window shown in Figure 6-9, specify a Servlet Engine Name. In the Servlet Engine name field, enter DomServlet Engine. Click **OK**.

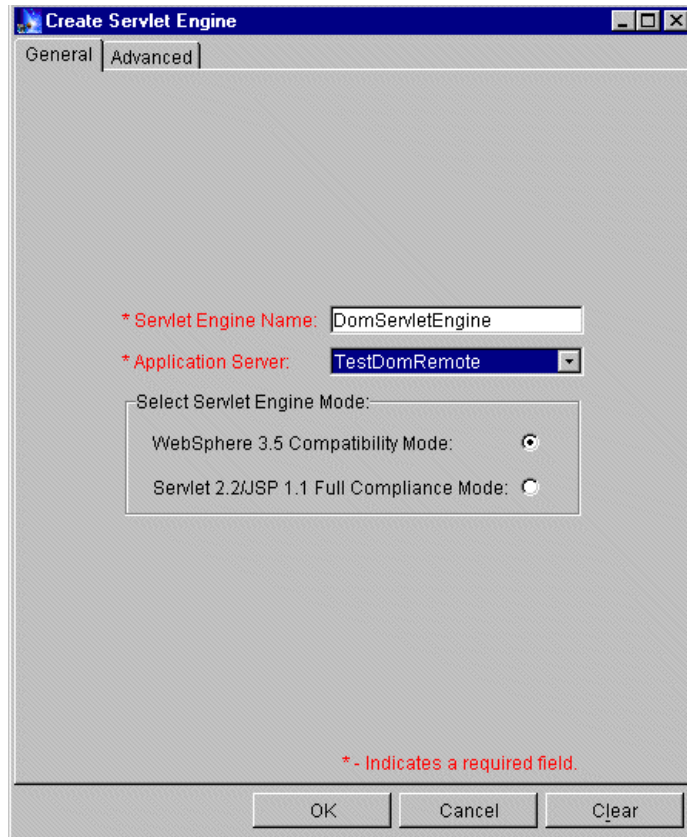


Figure 6-9 Create Servlet Engine window

If your Servlet Engine is successfully created, the pop-up window shown in Figure 6-10 appears. Click **OK**.



Figure 6-10 Create Servlet Engine success message

6. Now that you have successfully created a Servlet Engine in your WebSphere Application Server, you must create a Web application under your Servlet Engine. From the WebSphere Administrative Console, expand your application server, right-click your Servlet Engine, select **Create...**, and then select **Web Application** as shown in Figure 6-11.

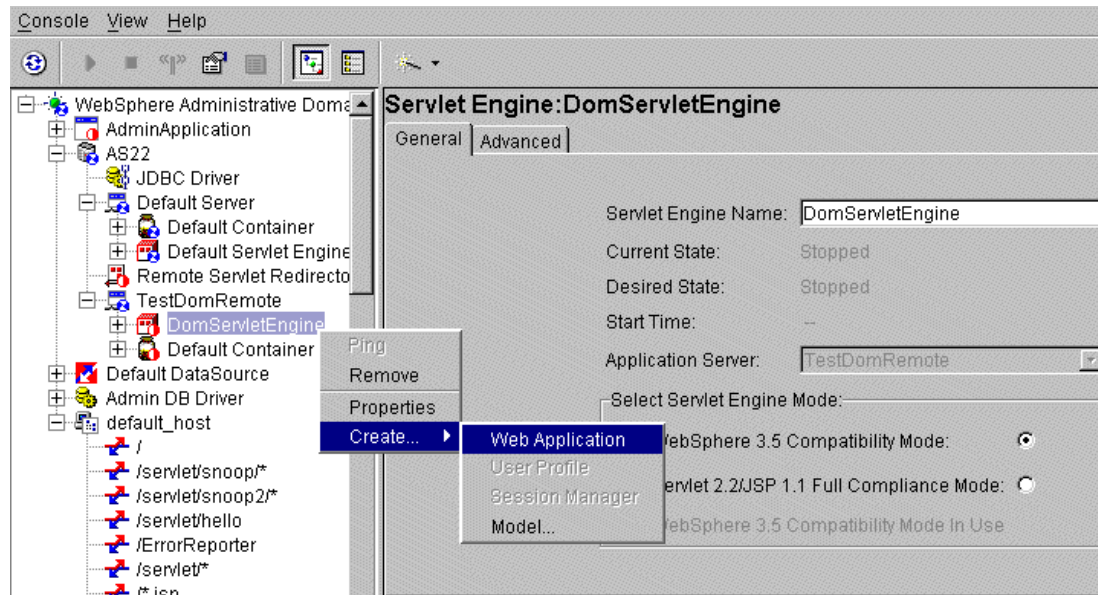


Figure 6-11 Create Web application

7. On the Create Web Application window, shown in Figure 6-12, enter a Web application name. In our example, we entered TestDom. Optionally, you can also enter a description for your Web application. Click **OK**.

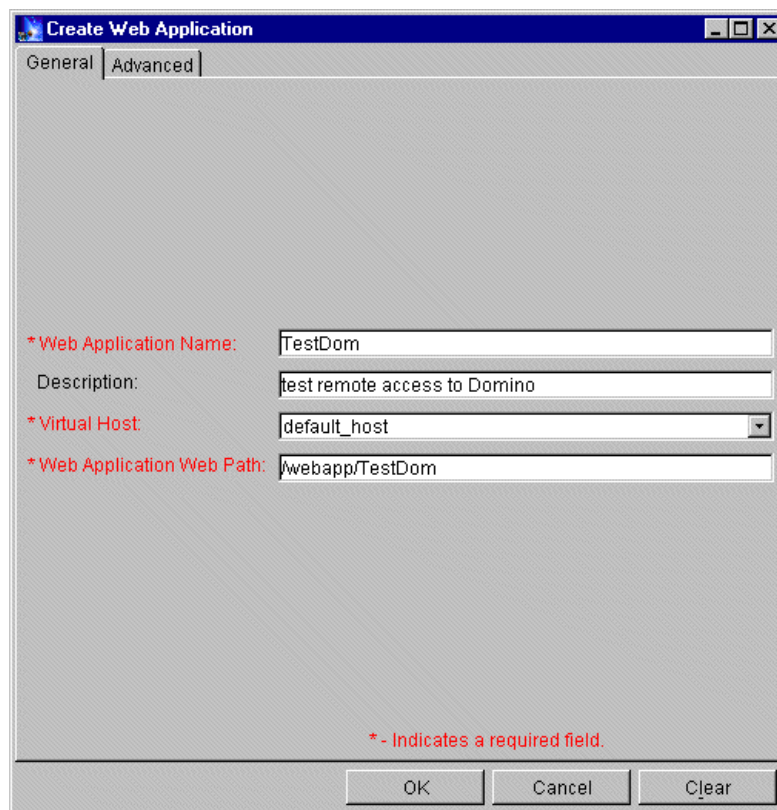


Figure 6-12 Create Web Application window

If your Web application is successfully created, the pop-up window shown in Figure 6-13 appears. Click **OK**.

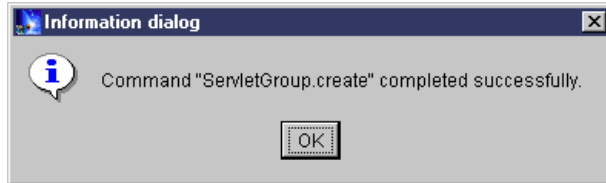


Figure 6-13 Create Web application success message

8. You now create a servlet in your newly created Web application. From the WebSphere Administrative Console, right-click the Web application and select **Create...**, and then select **Servlet** as shown in Figure 6-14.

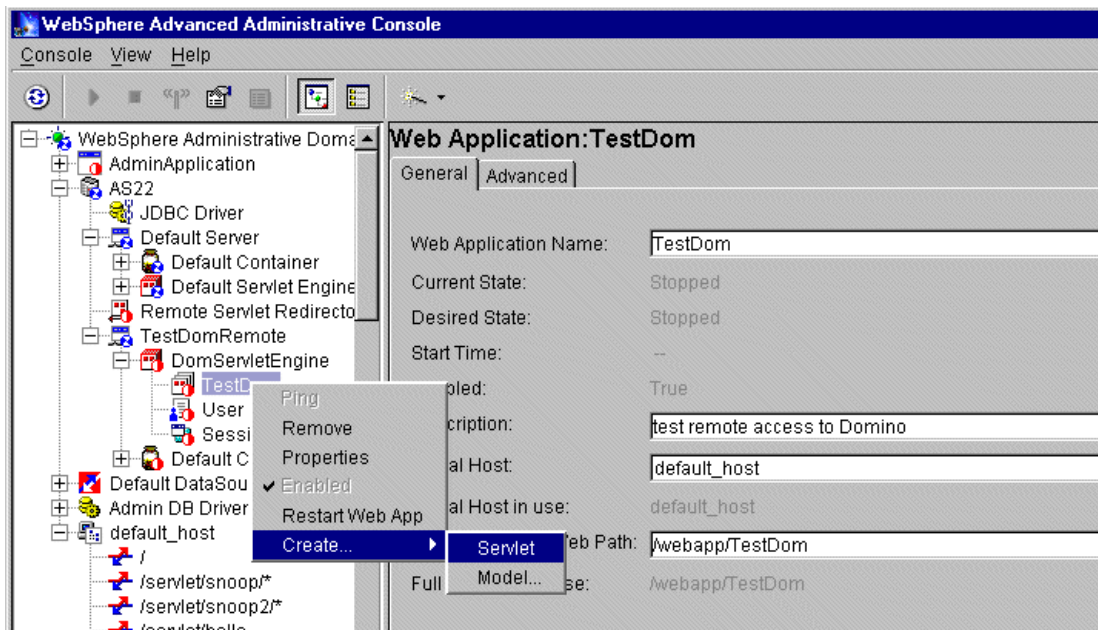


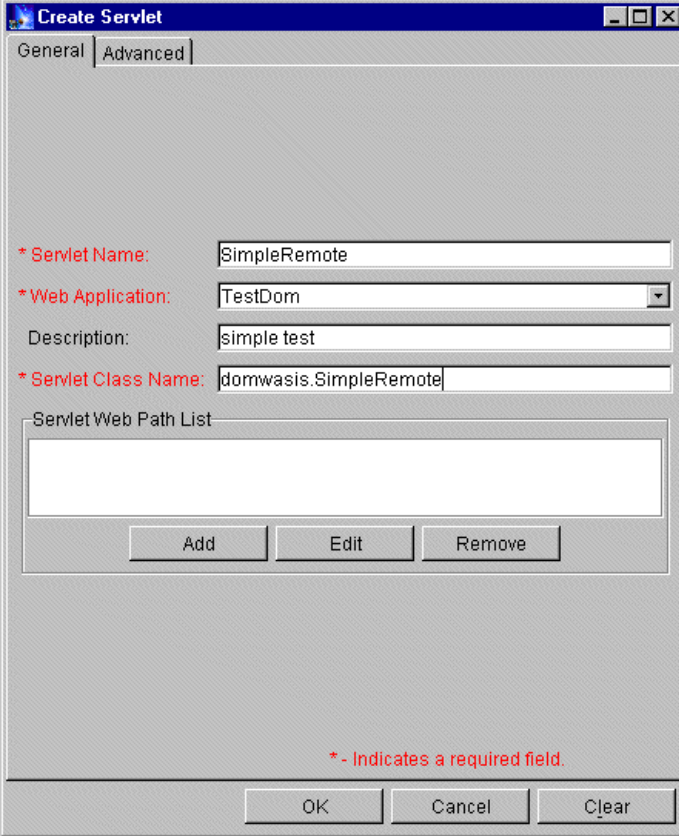
Figure 6-14 Create Servlet

9. On the Create Servlet window shown in Figure 6-15, specify the following parameters:

- Servlet Name: SimpleRemote
- Description: simple test
- Servlet Class Name: domwas1s.SimpleRemote

Click **Add**.





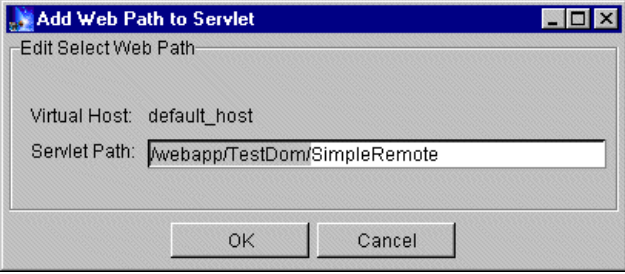
The 'Create Servlet' dialog box has two tabs: 'General' and 'Advanced'. The 'General' tab is active. It contains the following fields:

- \* Servlet Name: SimpleRemote
- \* Web Application: TestDom (dropdown menu)
- Description: simple test
- \* Servlet Class Name: domwasis.SimpleRemote

Below these fields is a 'Servlet Web Path List' section with an empty list box and three buttons: 'Add', 'Edit', and 'Remove'. At the bottom right, a red note states: '\* - Indicates a required field.' At the very bottom are 'OK', 'Cancel', and 'Clear' buttons.

Figure 6-15 Create Servlet window

10. On the Add Web Path to Servlet pop-up window (see Figure 6-16) type SimpleRemote at the end of the Servlet Path field and click **OK**.



The 'Add Web Path to Servlet' dialog box has a single tab labeled 'Edit Select Web Path'. It contains the following fields:

- Virtual Host: default\_host
- Servlet Path: /webapp/TestDom/SimpleRemote

At the bottom are 'OK' and 'Cancel' buttons.

Figure 6-16 Add Web Path to Servlet pop-up window

11. You return to the Create Servlet window, which automatically fills in the Servlet Web Path List field as shown in Figure 6-17. Click **OK** to create.



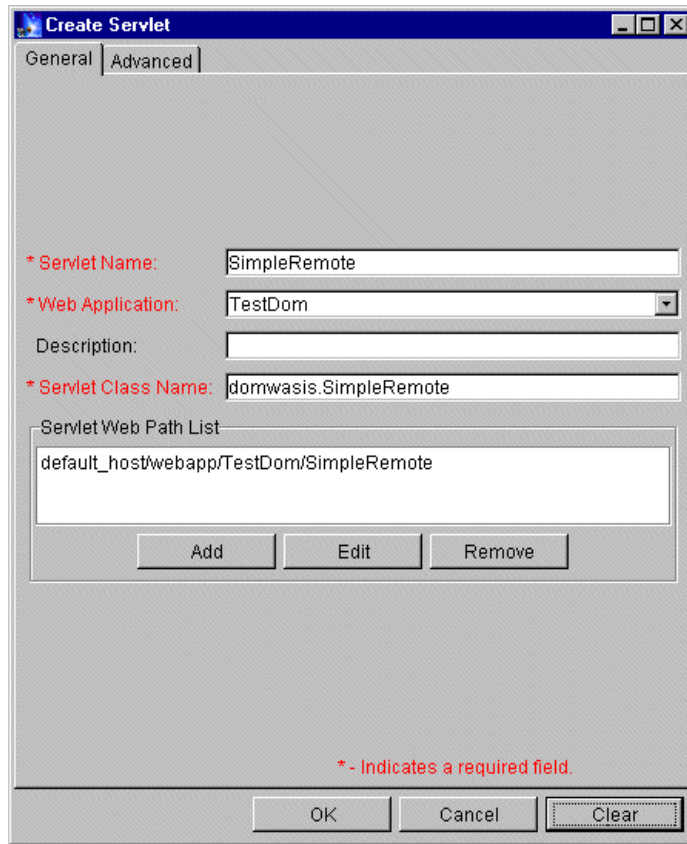


Figure 6-17 Create Servlet window

12. If your Servlet is successfully created, the pop-up window shown in Figure 6-18 appears. Click **OK**.



Figure 6-18 Create Servlet success message

13. You are now ready to start your WebSphere Application Server. Right-click the **TestDomRemote** Application Server. Click **Start** as shown in Figure 6-19.

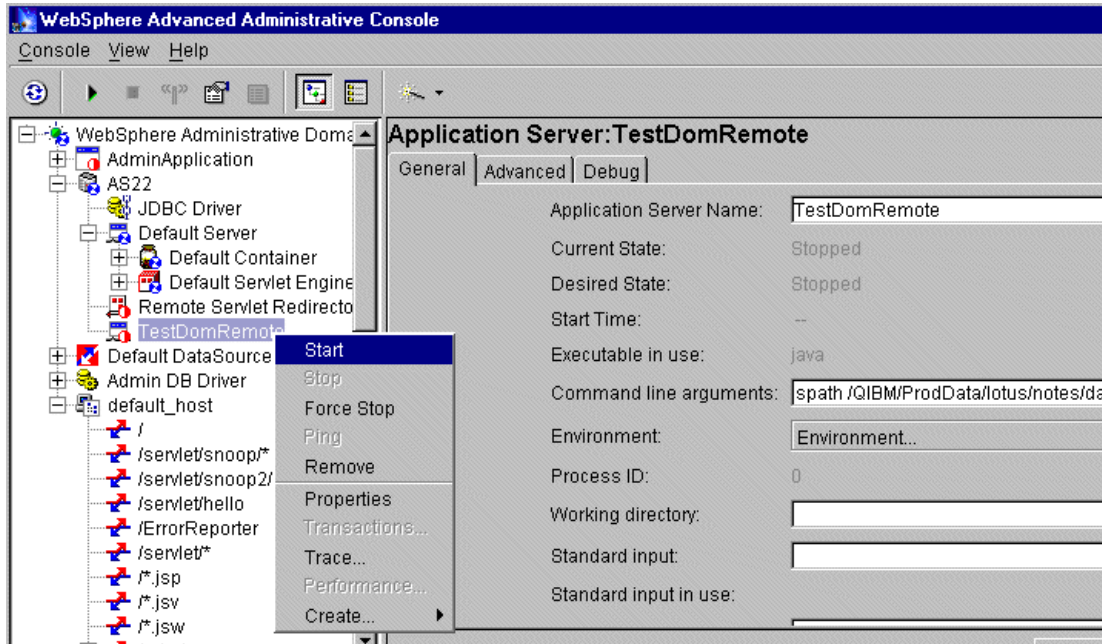


Figure 6-19 Starting the Application Server

14. If your WebSphere Application Server starts successfully, the pop-up window shown in Figure 6-20 appears. Click **OK**.

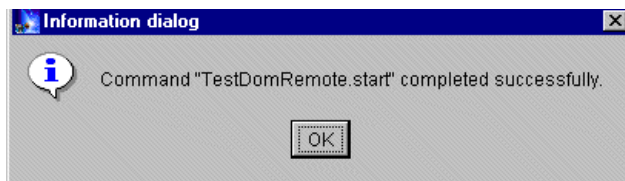


Figure 6-20 WebSphere Application Server started successfully message

15. Next we configure the default host settings. Right-click **default\_host** and select **Properties** as shown in Figure 6-21. The default host Properties window opens (see Figure 6-22).

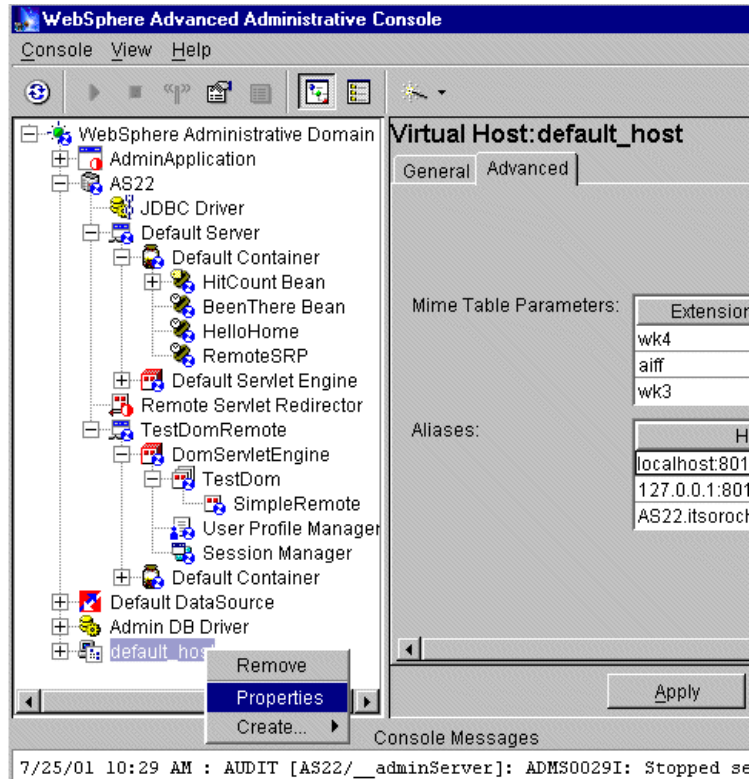


Figure 6-21 Default host menu

16. From the default\_host Properties window, select the **Advanced** tab. In the section labeled Aliases, enter each of the aliases by which your HTTP server (in this case, your Domino server) is known as illustrated in Figure 6-22. The list of aliases should include the host name of the Domino server and its IP address. In this example we specified each alias with the port number appended to the end, such as AS22.roch.ibm.com:8011 because we used a different port from the default HTTP port 80 for the Domino server. If you configure port 80 to receive HTTP requests, you don't need to specify the port number with the alias. After you have entered the host alias, click **Apply**.

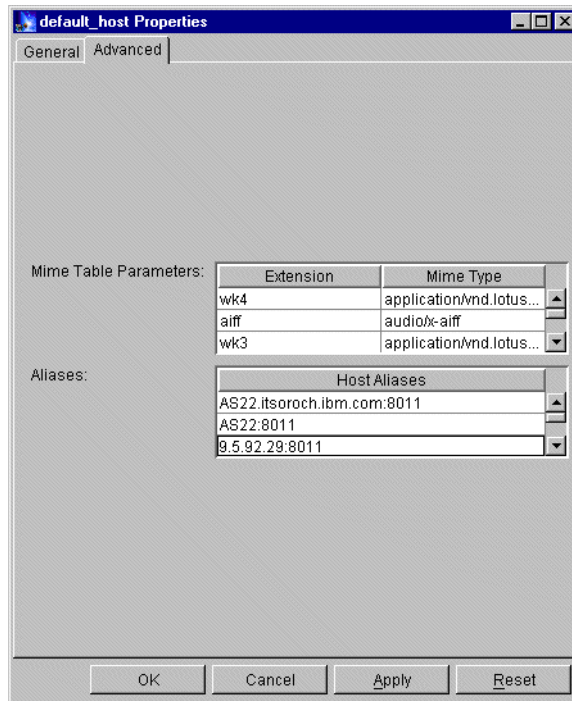


Figure 6-22 Specifying the virtual host aliases

17. To test the servlet, open a Web browser. On the URL line, enter the path to the servlet. This consists of the host name or one of its aliases, the port number if different from port 80, and the application path as you have defined it in the WebSphere Administration Console. In the example, the path to the servlet is:

`http://domwas11:8011/webapp/TestDom/SimpleRemote`

18. The Servlet output appears as shown in Figure 6-23.

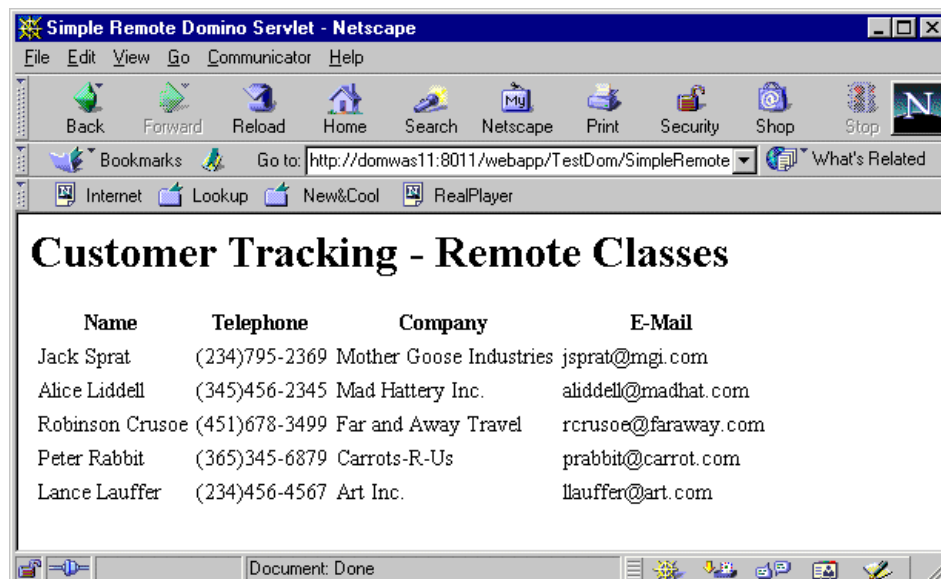


Figure 6-23 Output from SimpleRemote servlet

## More examples

Another example included in the download materials is an expansion on the TestDom application. This example consists of a servlet, `DisplayCustInfoServlet`, a `JavaBean`, `CustInfoBean`, `JavaServer Pages (JSP)`, `CustDisplay.jsp`, `GetCustomer.html`, and the `CustTrack.nsf`.

In the example, a user fills in an HTML-based search form by making a selection from a list of customer names. When the user submits the form, a servlet runs that searches the database for a document where the customer name matches the one specified by the user. Once the servlet locates the document, it extracts the name, telephone number, company and e-mail address and instantiates a `JavaBean` and places the information into it. The servlet calls a JSP that extracts information from the bean and displays it as a page in the user's Web browser. For information on how to download this example, refer to Appendix B, "Additional material" on page 295. To view the code listings for this example, refer to Appendix A, "Code examples" on page 265.

### 6.3.2 Configuring access to Domino using the local classes

To access Domino using the local classes, you would use the `Notes.jar` file instead of `NCSOW.jar`. Domino must be installed on the same machine on which the servlet is running. There are some additional configuration steps that are required.

In this section we configure a sample application that uses the local Domino objects. To complete our example, you need to have downloaded the following files:

- ▶ `CustTrack.nsf`
- ▶ `SimpleLocal.java`

Refer to the download instructions in Appendix B, "Additional material" on page 295 for information on how to obtain the files. `SimpleLocal.java` is found in the repository file included with the download materials. Follow the instructions in the appendix for importing the `VisualAge` for Java repository file.

#### Local example: Before you start

Prior to following the procedure for configuring the WebSphere Application Server, perform the following steps to configure the Domino environment and to modify and compile the servlet using `VisualAge` for Java:

1. Place the `CustTrack.nsf` file into the `Domino\Data` directory of your Domino server. Change the ownership of the file to the `QNOTES` user profile, entering the `CHGOWN` command on the OS/400 command line. In the example the command is:

```
CHGOWN OBJ ('/DOMINO/DOMWAS11/CUSTTRACK.NSF') NEWOWN(QNOTES)
```

2. Export the `SimpleLocal` class file from `VisualAge` for Java to `/QIBM/UserData/WebAsAdv/WASSERVER/default_host/TestDomLocal/servlets`.

In this directory, replace `WASSERVER` with the name of your WebSphere Application Server. For information on how to export from `VisualAge` for Java to a directory, refer to the `VisualAge` for Java product documentation.

For your reference, a full code listing of the `SimpleLocal` servlet is also available in Appendix A, "Code examples" on page 265.

#### Configuring the SimpleLocal servlet

Figure 6-24 illustrates the WebSphere Application Server topology for the local Domino objects example.

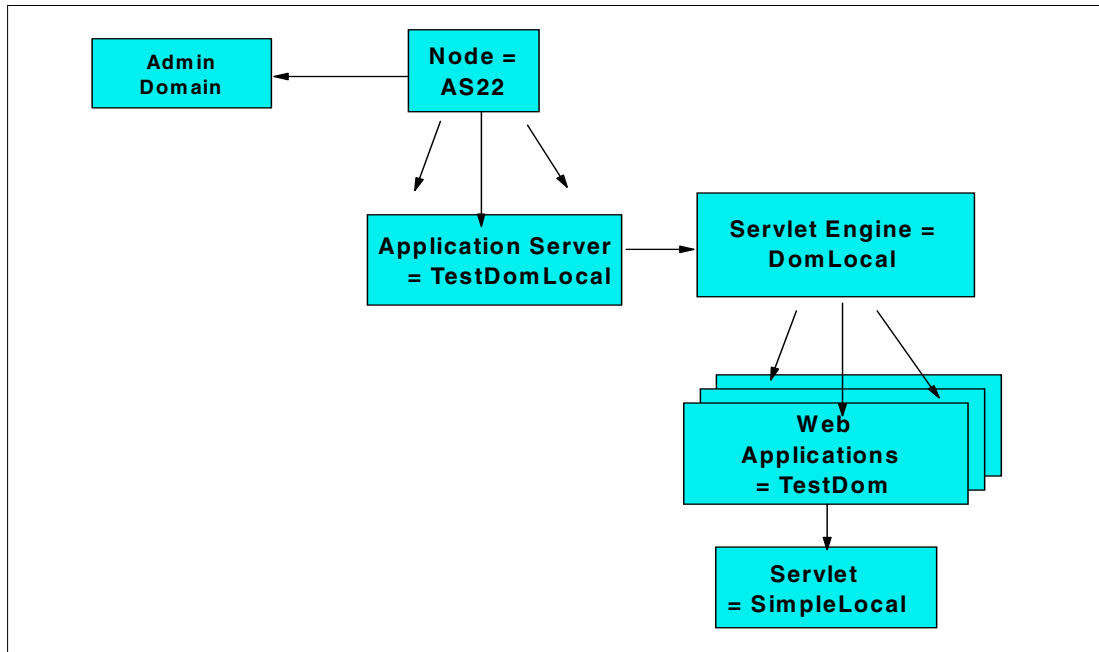


Figure 6-24 WebSphere Application Server topology for Local Domino access example

1. When you use the local Domino objects, the WebSphere Application Server must be run using the QNOTES user ID. You must use OS/400 Operations Navigator to give the user ID QNOTES the authority to run WebSphere.

Start Operations Navigator and right-click your iSeries server. Select **Application Administration** as shown in Figure 6-25.

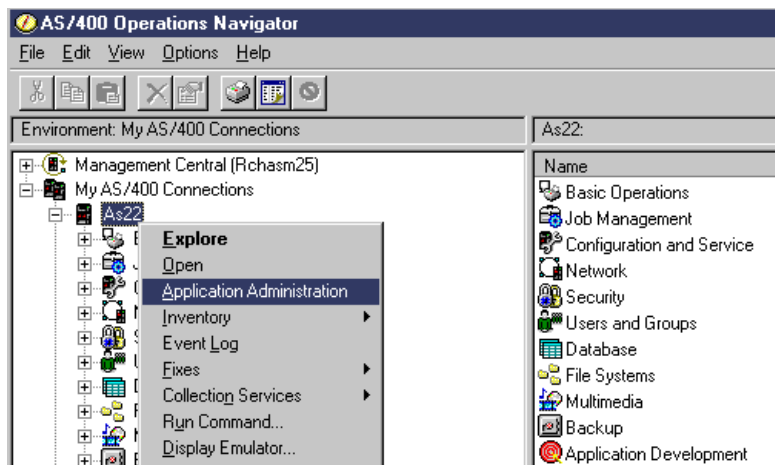


Figure 6-25 Operations Navigator

2. The Applications window shown in Figure 6-26 appears. Click **OK** to continue.

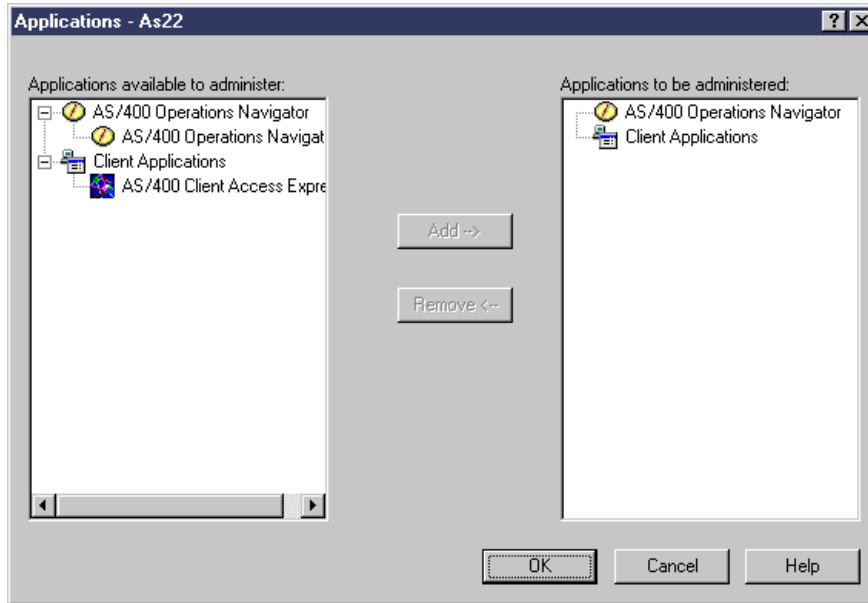


Figure 6-26 Operations Navigator: Applications window

- On the Application Administration window that appears, select the **Host Applications** tab. Expand the **QIBM\_EJB\_PRODUCT** function and then expand the **QIBM\_EJB\_GROUP\_OF\_FUNCS**. Highlight the **QIBM\_EJB\_SERVER\_FUNC** function as shown in Figure 6-27 and click the **Customize** button at the bottom of the window.

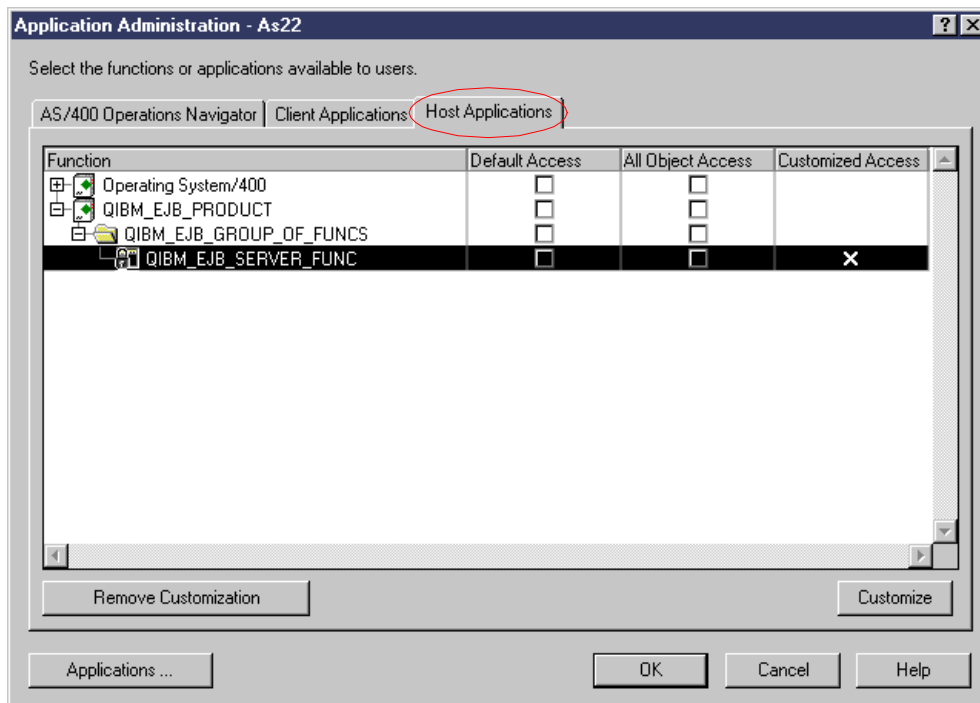


Figure 6-27 Operations Navigator Application Administration

- On the Customize Access window that appears, expand the **All Users** list in the Users and groups area. Select the user of QNOTES and click the **Add** button to add the QNOTES User ID profile to the Access allowed area, as shown in Figure 6-28. Click **OK** twice to return to the main Operations Navigator window.

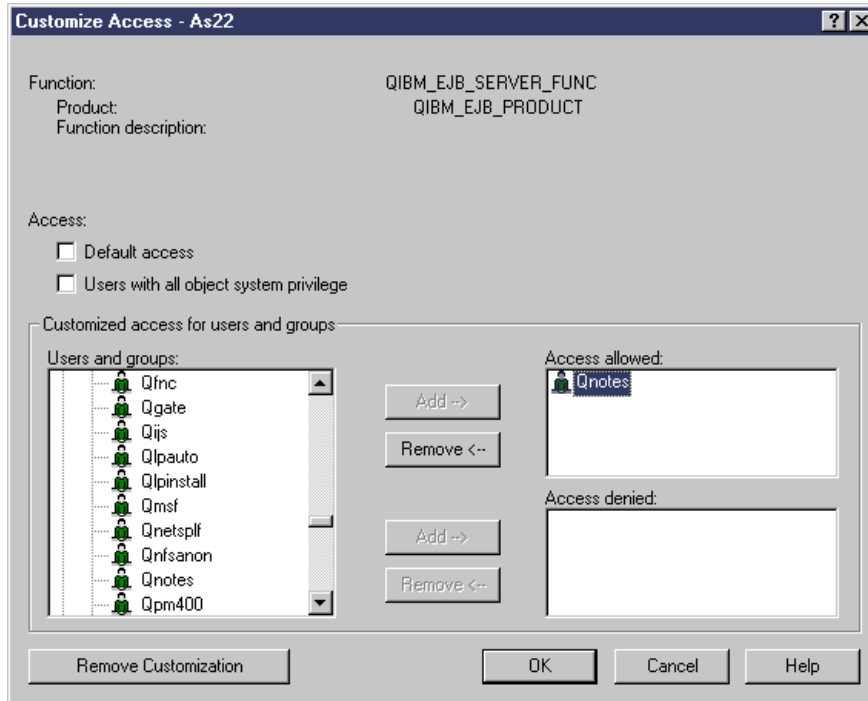


Figure 6-28 Operations Navigator Customize Access

5. You now need to give the QNOTES user profile read/write (\*RW) authority to the files stdout.txt and stderr.txt in OS/400 Integrated File System (IFS) directory of /QIBM/UserData/WebASAdv/default.

These files are used by the WebSphere Application Server to specify the standard output stream and standard error stream for OS/400, which then defaults to QPRINT.

From an OS/400 command line, execute the Change Authority (CHGAUT) command with the following parameters:

```
CHGAUT OBJ('/qibm/userdata/webasadv/default/stdout.txt') DTAAUT(*RW)
CHGAUT OBJ('/qibm/userdata/webasadv/default/stderr.txt') DTAAUT(*RW)
```

**Note:** You must make sure that the files stdout.txt and stderr.txt exist in the IFS directory of /QIBM/UserData/WebASAdv/default. You can check by using either the Work Link (WRKLNK) command from a 5250 emulation window or you can use Operations Navigator. In Operations Navigator, you must expand the File Systems view, then expand the Integrated File System view and finally the Root view to see the QIBM directory.

If these text files do not exist, you can create them on your PC workstation by using a text editor such as NotePad. The files should be empty, but make sure they have an extension of .txt. To copy these files to the OS/400 IFS directory, you can map a drive from your PC workstation and copy them or you could FTP them to the IFS directory.

6. You also must add the QNOTES library to the default user library list. Copy one of the job descriptions in library QUSRNotes (WRKJOBQD QUSRNOTES/\*ALL), such as DOMINO00 or DOMINO01, to something like QNOTESJOBQD. This job description contains a library list because it contains the QNOTES library. Now change the QNOTES user profile to use QUSRNOTES/QNOTESJOBQD as its default job description instead of the standard QDFTJOBQD. Now when WebSphere switches to user profile QNOTES, it will get QNOTES in the library list via this job description.



7. Now you are ready to create an Application Server in WebSphere. In the WebSphere Administrative Console window, right-click the node, as shown in Figure 6-7 on page 159. Select **Create...** and then select **Application Server**. The Create Application Server window appears as shown in Figure 6-29.

The screenshot shows the 'Create Application Server' dialog box with the following fields and values:

- \* Application Server Name:** (empty text box)
- Command line arguments:** -Xms32m
- Environment:** Environment...
- Working directory:** (empty text box)
- Standard input:** (empty text box)
- Standard output:** stdout.txt
- Standard error:** stderr.txt
- Maximum startup attempts:** 2

A red asterisk at the bottom indicates a required field. Buttons at the bottom are OK, Cancel, and Clear.

Figure 6-29 Create Application Server window

8. In the Application Server Name field, enter TestDomLocal.

9. In the Command line arguments field, enter:

-Xms32m -classpath /QIBM/ProdData/Lotus/Notes/Notes.jar

10. In the Standard output field, enter:

/qibm/userdata/webasadv/default/stdout.txt

11. In the Standard error field, enter:

/qibm/userdata/webasadv/default/stderr.txt

The completed General tab of the Create Application Server window appears as shown in Figure 6-30.

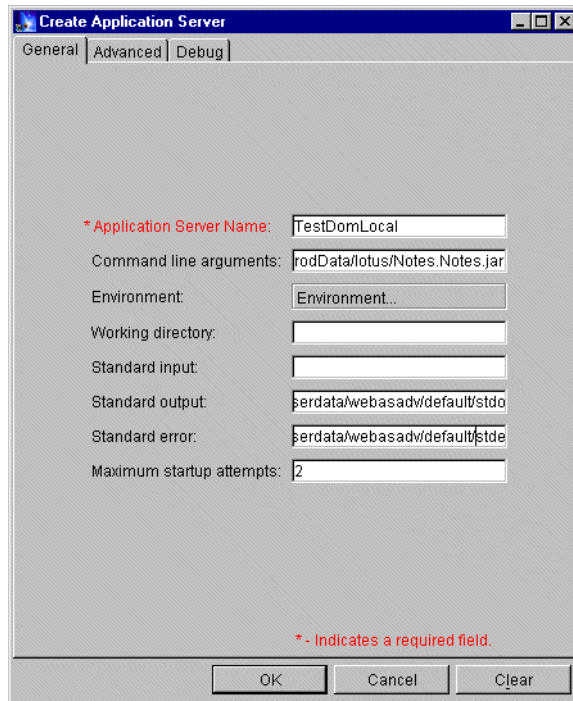


Figure 6-30 Create Application Server window filled in

12. Click the **Environment** button located in the middle of the window. This allows you to specify environment variables, and their values, to be used by the application server. The Property Editor Environment Editor window opens.
13. In the Property Editor Environment Editor window, shown in Figure 6-31, enter PATH for the Variable Name field and the following for the Value field:

/domino/dominoserver:/qibm/userdata/lotus/notes:/qibm/proddata/lotus/notes

Where /domino/dominoserver is the data directory of your iSeries Domino server. In this example we used /domino/domwas11, which is the data directory of our iSeries Domino server called "domwas11". Click **Add**, then click **OK**.

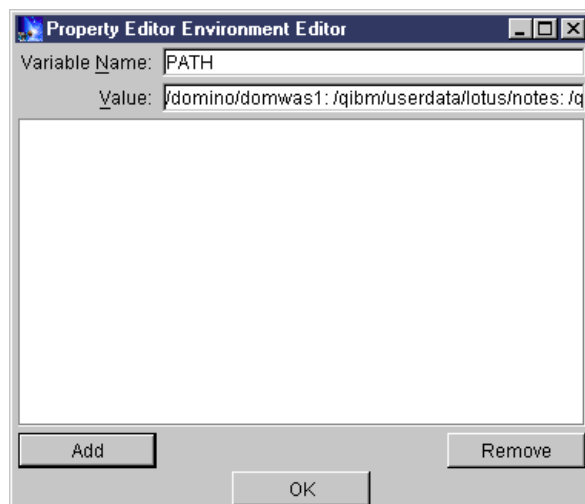


Figure 6-31 Property Editor Environment Editor

14. In the Create Application Server window, click the **Advanced** tab. Specify `QNOTES` in the User ID field as shown in Figure 6-32. This specifies the user profile under which WebSphere runs. Click **OK**.

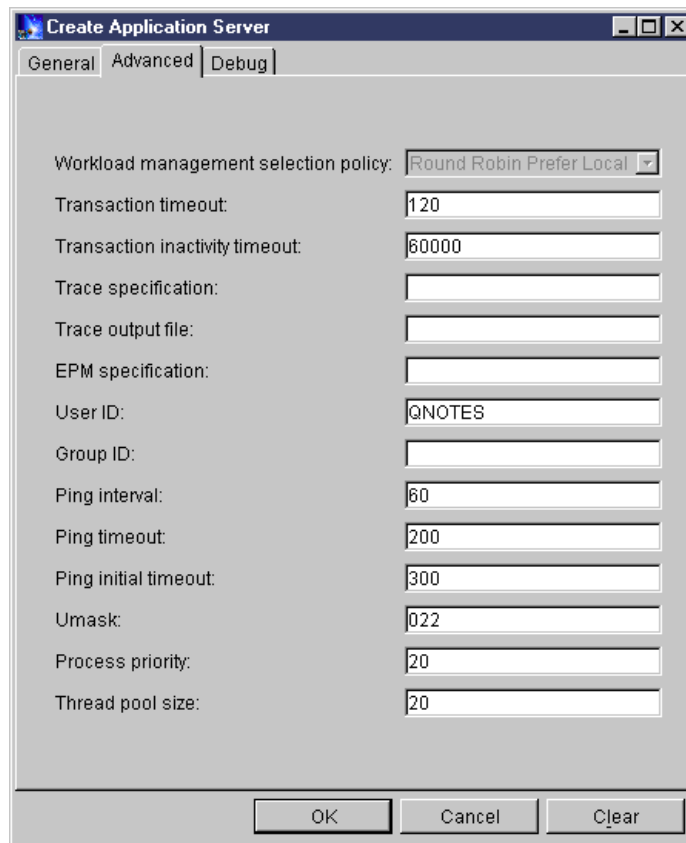


Figure 6-32 Create Application Server, Advanced tab

15. The pop-up window shown in Figure 6-33 appears if your WebSphere Application Server was created successfully. Click **OK**.

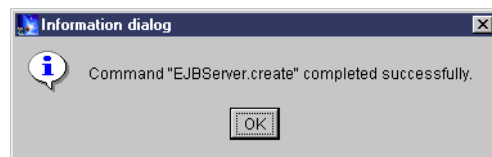


Figure 6-33 Create WebSphere Application Server success message

You should also see your newly created WebSphere Application Server listed under your node in the WebSphere Administrative Console.

16. Create the Servlet Engine. Right-click the application server and choose **Create...** and then **Servlet Engine**. The Create Servlet Engine window opens.

In the Servlet Engine Name field, specify `DomLocal1`. In the Application Server field, select **TestDomLocal** from the list. The completed window should appear as shown in Figure 6-34. Click **OK**.

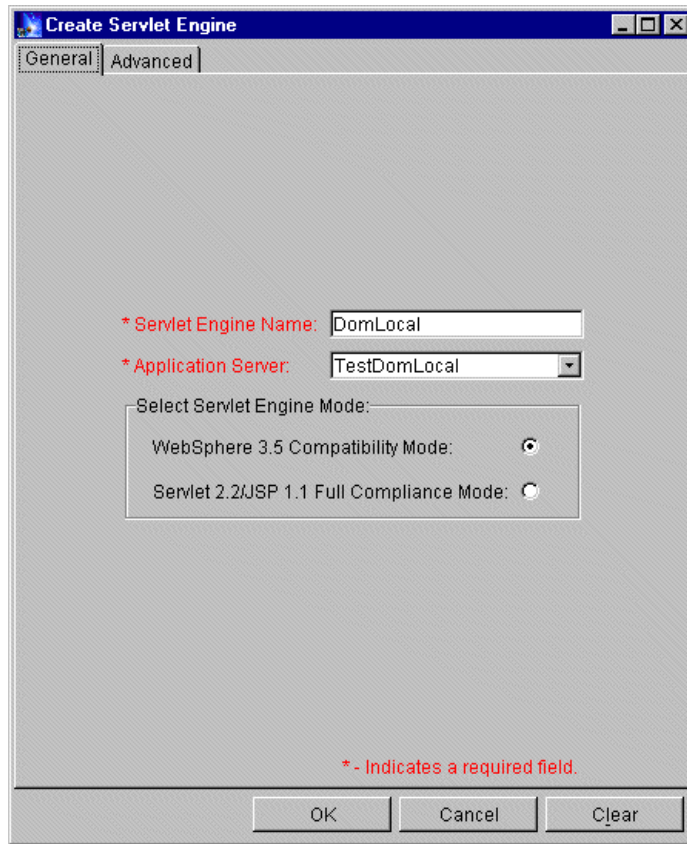


Figure 6-34 Create Servlet Engine

17. A message confirming the successful creation of the Servlet Engine appears as shown in Figure 6-10 on page 161. Click **OK**.
18. Create a Web application. Right-click the Servlet Engine and select **Create...** and then **Web Application** as shown in Figure 6-35. The Create Web Application window opens.

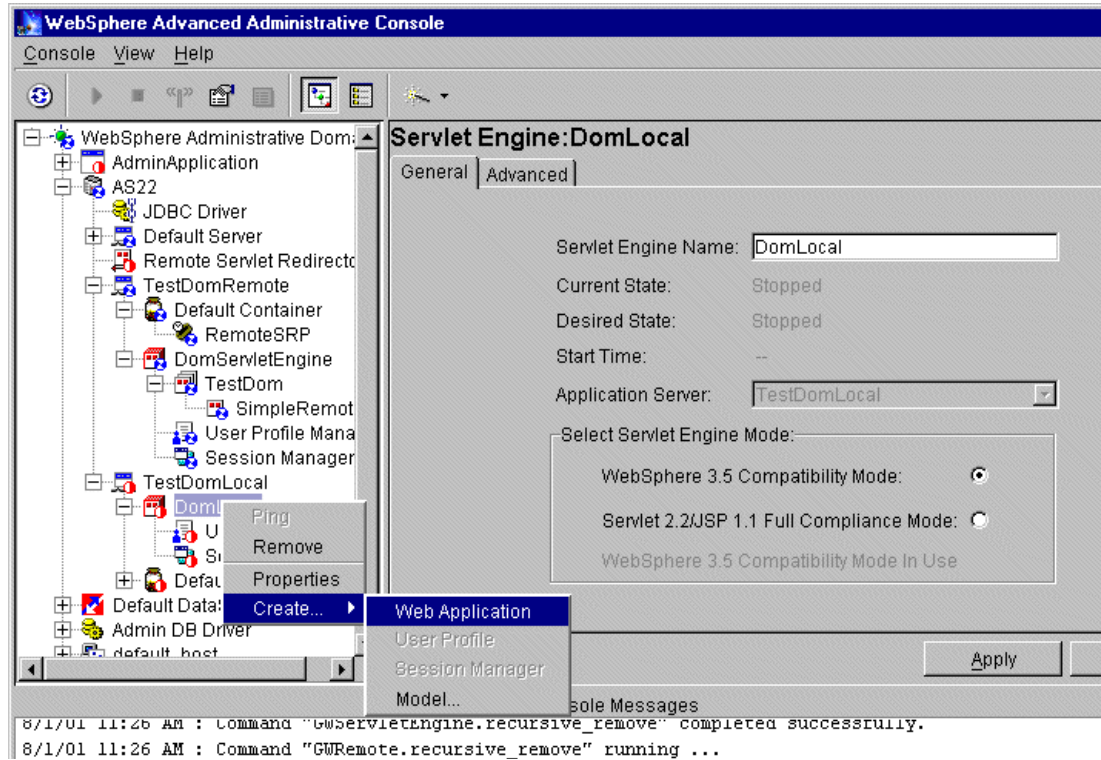


Figure 6-35 Create Web application

19. In the Web Application Name field, enter TestDomLocal. The completed window should appear as illustrated in Figure 6-36. Click **OK**.

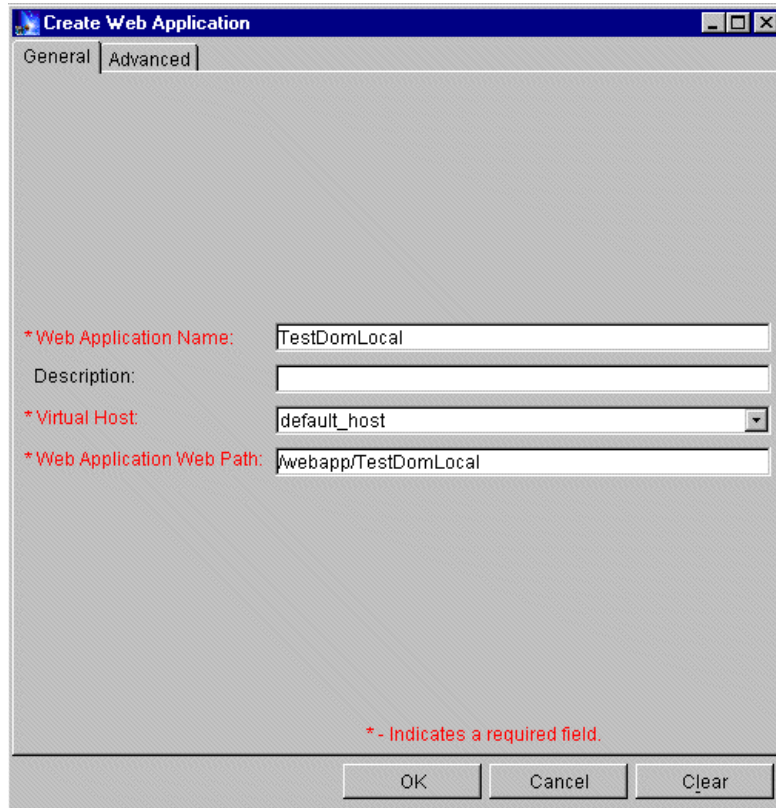


Figure 6-36 Create Web Application

20. If the Web application has been created successfully a window like the one shown in Figure 6-37 appears. Click **OK**.

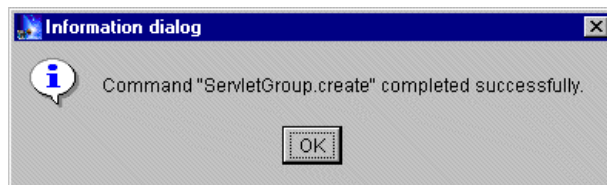


Figure 6-37 Create Web application success message

21. You now create a servlet in your newly created Web application. From the WebSphere Administrative Console, right-click the Web application, select **Create...** and then select **Servlet** as shown in Figure 6-38.

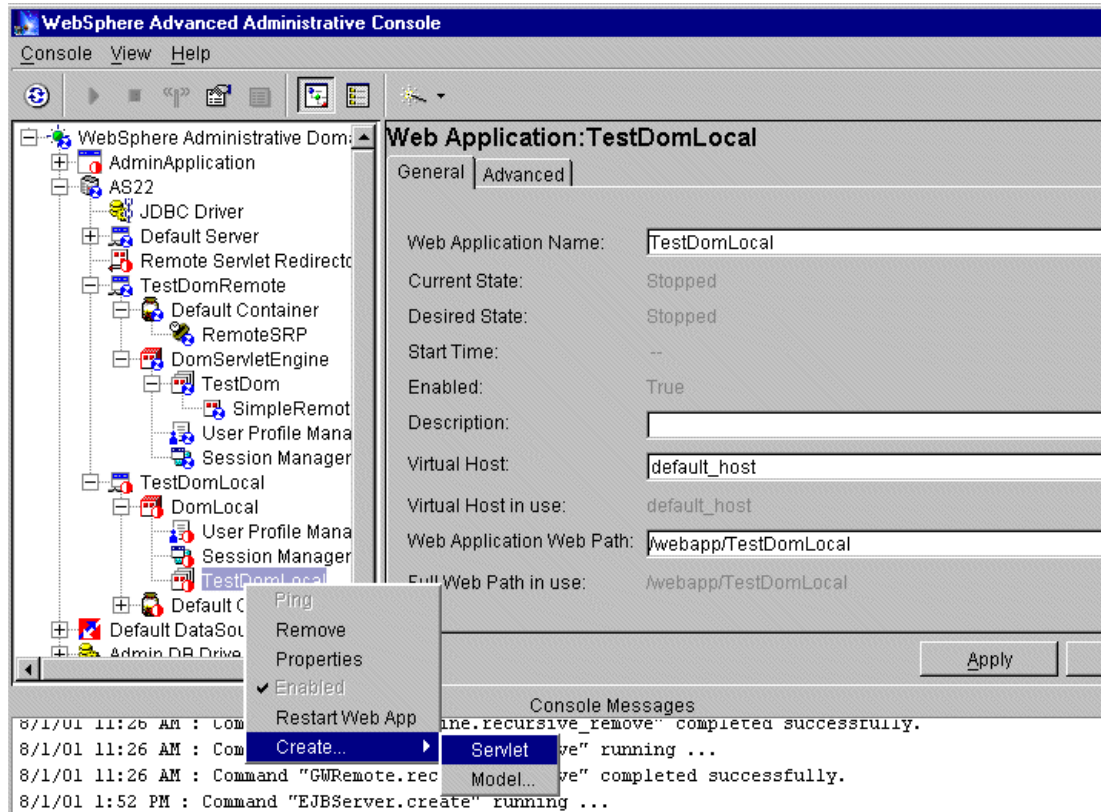


Figure 6-38 Creating a servlet from the WebSphere Administrative Console

22. On the Create Servlet window shown in Figure 6-39, specify the following parameters:

- Servlet Name: SimpleLocal
- Description: simple test of local access to Domino
- Servlet Class Name: domwasis.SimpleLocal

Click **Add**.

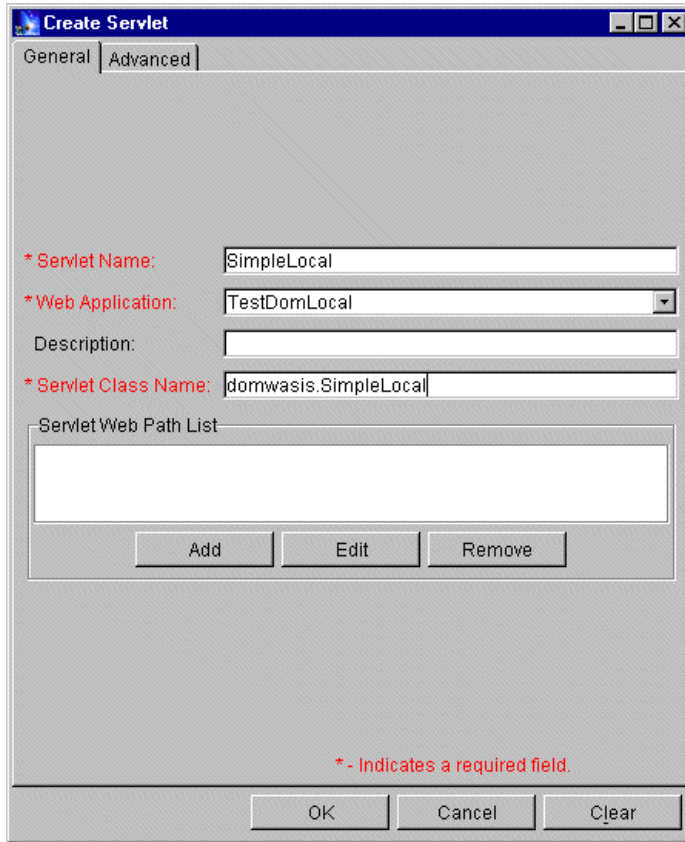


Figure 6-39 Create Servlet window

23. In the Add Web Path to Servlet window that opens, type SimpleLocal into the Web Path field as shown in Figure 6-40. Click **OK** to return to the Create Servlet window and click **OK** again to close the Create Servlet window.

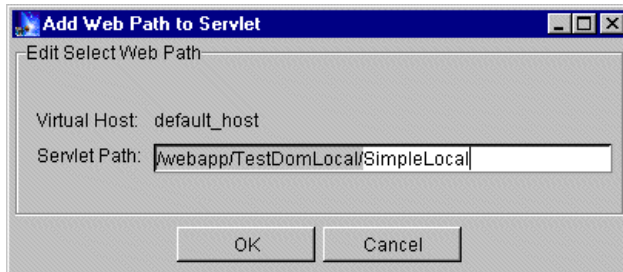


Figure 6-40 Add Web Path to Servlet window

24. Next, start the Application Server by right-clicking the application server name and selecting **Start** as shown in Figure 6-41.



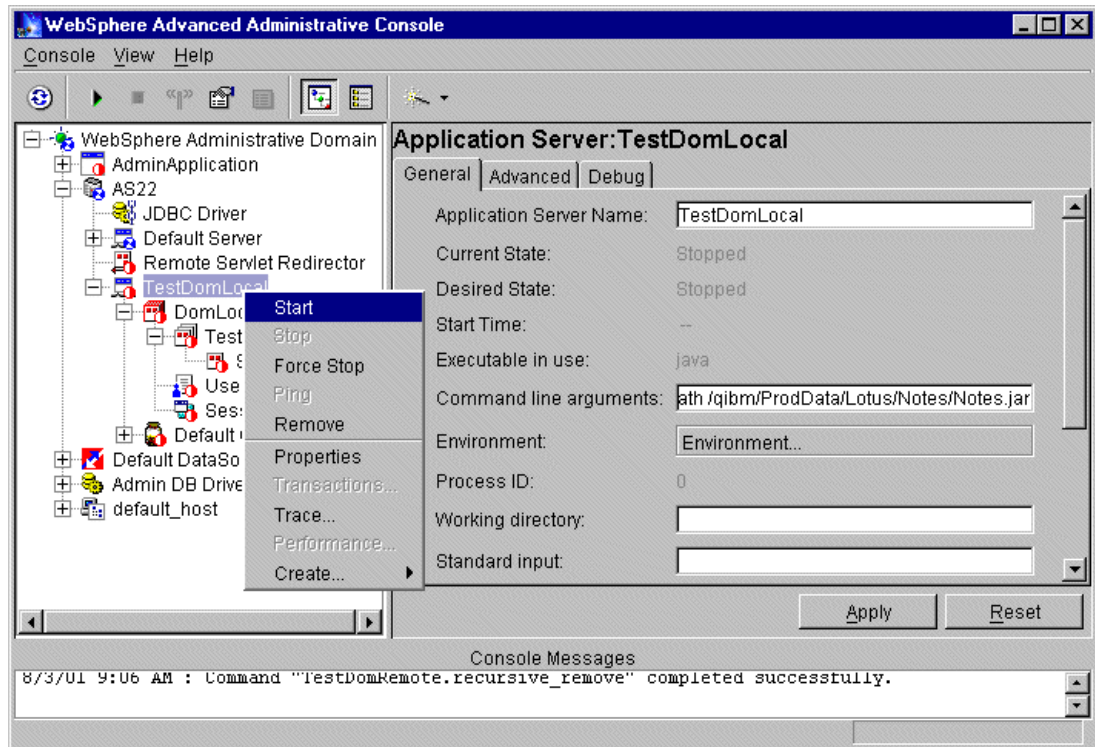


Figure 6-41 Starting Application Server window

25. To test the servlet, open a Web browser. On the URL line, enter the path to the servlet. This consists of the host name or one of its aliases, the port number if different from port 80, and the application path as you have defined it in the WebSphere Administration Console. In our example, the path to the servlet is  
<http://domwas11:8011/webapp/TestDomLocal/SimpleLocal>.

26. The Servlet output appears as shown in Figure 6-42.

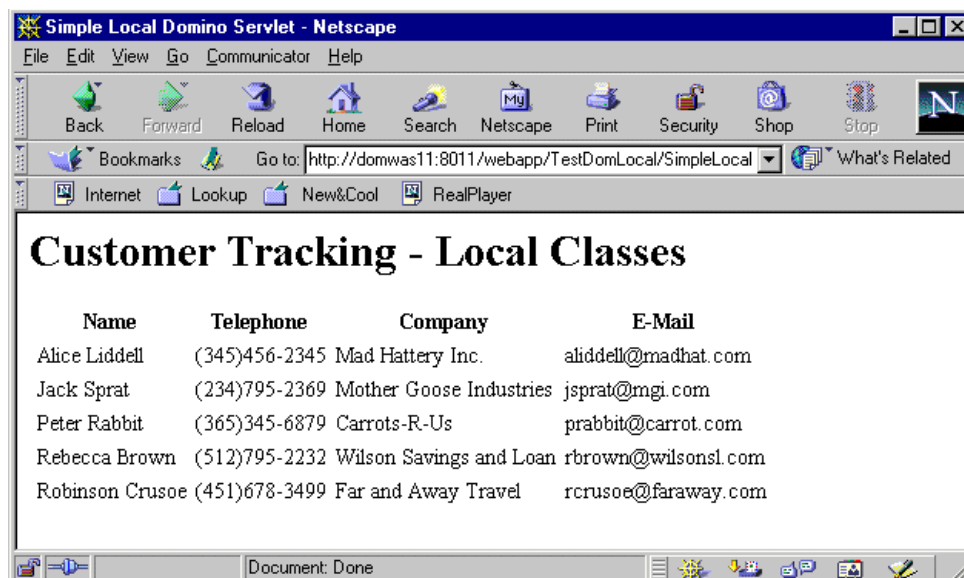


Figure 6-42 Output from SimpleLocal servlet

### 6.3.3 Performance of local versus remote access

The decision between using the local classes or the remote ones involves making trade-offs in the architectural design of your system. Using local objects requires that you have sufficient CPU capacity to support Domino and WebSphere running on the same machine. Another consideration is that local Domino objects cannot be persisted across servlet invocations because of the single thread restriction. Using remote objects, Domino and WebSphere can run on separate machines; however, the speed of the network can affect performance. In our testing of local versus remote access, we have found remote access to be faster. However, you should note that in our test environment, Domino and WebSphere were running on the same machine, so while we were able to test the time required for processes needed by CORBA, we did not evaluate the effect of the network bandwidth on performance.

The remote access obtains a handle to a Notes session object once for all executions to a servlet because the remote Domino objects can be accessed in multiple threads. The local access has to initialize and terminate the Notes environment on each call to a servlet because local objects are accessed by only one thread. Of course, individual results may vary depending upon the specific coding in your application. For tips on how you can optimize your code, we recommend that you refer to the *Lotus Domino & Notes R5 Best Practices Guide*, which can be obtained at: <http://www.lotus.com/partners/bpg.nsf>

## 6.4 Application development with single sign-on

In Domino 5.05 the Java API was enhanced to support single sign-on (SSO) between Domino and WebSphere. In other words, a user would initially be authenticated by either Domino or WebSphere Application Server. Web users can log on once to a Domino or WebSphere server, then access any other Domino or WebSphere server in the same DNS domain that is enabled for single sign-on without logging on again.

When the user performs an action that accesses a resource on another server within the single sign-on domain, a session token is passed to the other server so that the user may be authenticated there as well. For information on how to configure single sign-on, refer to “What is single sign-on?” on page 124. Enhancements to the programming API include the introduction of the `getSessionToken()` method of the `lotus.domino.Session` class and some extensions to the `NotesFactory` class. The following new methods were added the `NotesFactory` class:

- ▶ `createSession(hostString, String Token)`: Access is granted based on the token. This method works in a Domino environment. The token must be a valid token for single sign-on obtained from `Session.getSessionToken` or the `LtpaToken` cookie used by WebSphere.
- ▶ `createSession(hostString, org.omg.SecurityLevel2.Credentials)`: Access is based on the `Credentials` object. This method works in a WebSphere environment where the `Credentials` object is created using `loginHelper`.
- ▶ `CreateSession(hostString, null)`: Access is granted based on the current `Credentials` object in the WebSphere environment. This method works from an Enterprise JavaBeans (EJB) application in WebSphere.

The `getSessionToken()` method extracts a session token from the `lotus.domino.Session` object for the current user and stores it as a token in a format that is consistent with the `LtpaToken` cookie used by WebSphere. For more information on this topic, refer to the article entitled “Enabling Java API options for authenticated invocation” in the Domino R 5.05 Release Notes, found at: <http://www.notes.net/notesua.nsf/>



## Accessing Domino from WebSphere

This chapter describes several methods for creating WebSphere Enterprise Application components that access Domino for iSeries. A brief discussion covers how to set up the development environment. We discuss in detail how to build servlets, JSPs and Enterprise JavaBeans that integrate with Domino. Methods for testing and debugging are included in each section.

**Note:** All the examples used in this chapter were created with VisualAge for Java and are available for download from the IBM Redbooks Web site. See Appendix B, “Additional material” on page 295, for more information.

## 7.1 Development environment

In our example, we create a Web-based registration application for “Snow Dome World 2002”. People wishing to attend simply visit the Web site and complete the registration form. Those who have registered are able to return to the Web site and update their information as required.

Domino and WebSphere work together to produce the final application. The solution is based on Domino acting as the HTML and database server, while WebSphere provides the servlet and EJB runtime environment. The DSAPI filter as discussed in Section 3.2.1, “Configuring Domino HTTP server for use by WebSphere” on page 20, is used to integrate the two systems.

The application is developed in three sections. The first section is designed to show you how to use servlets. The second section introduces JavaServer Pages, and the third deals with Enterprise JavaBeans.

All examples in this chapter were developed and tested with the following software releases:

- ▶ For development:
  - VisualAge for Java 3.5.3 Enterprise Edition
  - An HTML editor such as WebSphere Studio 3.5
  - Lotus Domino Designer 5.0.7
- ▶ For deployment, our server environment included:
  - Lotus Domino Server 5.0.8 for iSeries
  - WebSphere Application Server V3.5.3 for iSeries
  - WebSphere Administration Client
  - WebSphere Studio 3.5

The installation and configuration of each of these products is not within the scope of this book. For detailed information on installation, refer to the product manuals.

As you work through this chapter, you can complete the code for each section as listed. A complete listing of all the code is available in Appendix A, “Code examples” on page 265. Details regarding downloading the complete source code can be found in Appendix B, “Additional material” on page 295.

### 7.1.1 Application overview

The Snow Dome application consists of 13 files. Table 7-1 lists the files and provides a brief overview of the role each plays.

*Table 7-1 Files needed for the Snow Dome application*

File	Role
SDIndex.html	Entry point for the application
regejb.html	Form for registering via EJB
registration.html	Form for servlet registration
regWJSP.html	Form for servlet registration that uses JSPs
snowdomebanner.gif	Image file
SDSearch.jsp	For locating registration documents

File	Role
SDAttendee.jsp	To display current registration information
SDSuccessUpdate.jsp	To display confirmation of details update, and new registrations
returnMessage.jsp	To display confirmation of registration via servlet
RegisterWJSP servlet	Servlet used for registration file regWJSP.html
Controller servlet	Used to control the logic flow of the application
Register servlet	Used to create registration document
registration.nsf	Notes database used to store records
RegDoc bean	Enterprise JavaBean used for writing and retrieving records from Domino

Figure 7-1 shows the site architecture. The examples in this chapter are designed to build the core of this application.

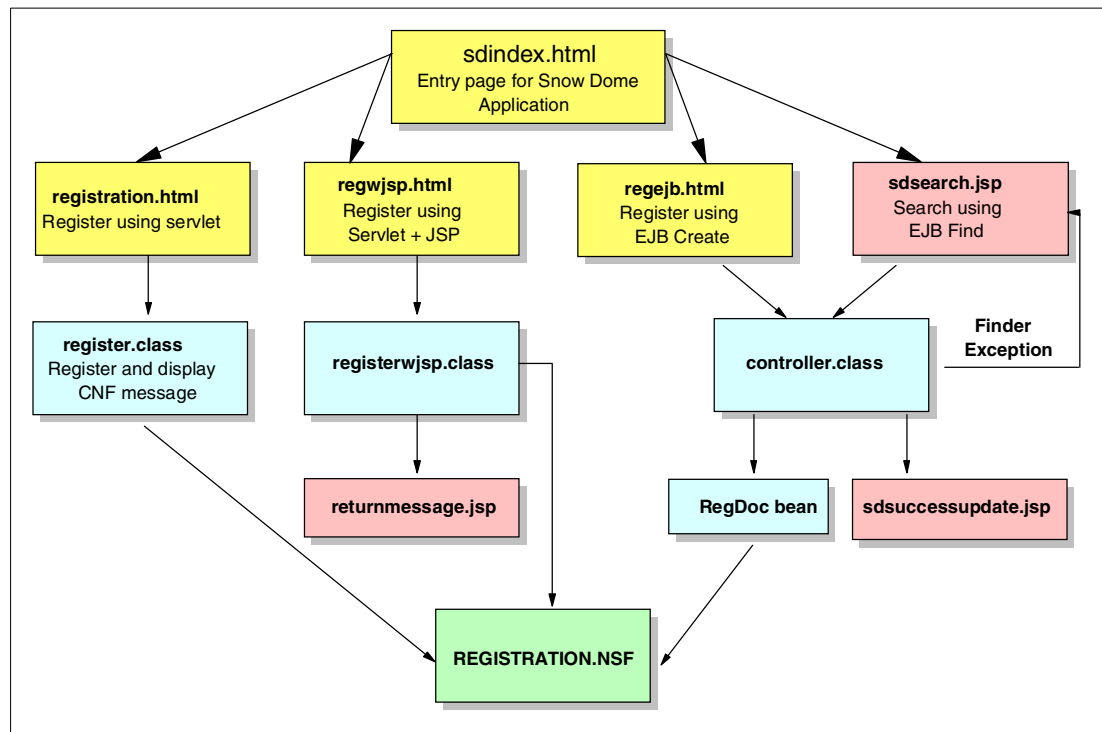


Figure 7-1 Site architecture

**Note:** In our example Domino is primarily used as a back-end store. Domino forms and views can also be used as Web windows. In our example we could easily change the HTML forms used for registration to be Domino forms displayed over the Web. The decision to use Domino forms or HTML forms should be based on the skills you have available. HTML forms can be developed quickly and with a variety of tools. HTML forms also have a slightly faster response time compared to Domino forms. Domino forms must be converted to HTML before being displayed. On the other hand, Domino forms allow you to include a rich set of features made available via Domino. These include (but are not limited to) field level security, encryption and multimedia support.

A complete listing of the source for each file is included in Appendix A, “Code examples” on page 265.

## 7.1.2 Setup of VisualAge for Java

Before we can begin building our application, we must prepare our development environment. The preparation includes:

- ▶ Customizing your VisualAge installation to include the required features
- ▶ Creating a project within VisualAge
- ▶ Creating packages within the project
- ▶ Modify the default\_app.webapp configuration file
- ▶ Creating a Domino database to store the registration information

This section shows how to customize your VisualAge for Java installation. We use VisualAge for Java for developing and testing all servlets and EJBs. Before completing this section you will need VisualAge for Java Enterprise Edition V3.5 installed on your workstation.

### Adding the required features

VisualAge includes many Java libraries that can be used in your development. These libraries are grouped together based on their function. A VisualAge feature is a collection of these libraries. When you add a feature to your VisualAge installation you are making a group of Java libraries available for use in your application.

1. Open VisualAge for Java by selecting **Start -> Programs -> IBM VisualAge for Java for Windows V3.5 -> IBM VisualAge for Java**.

You will be presented with the Welcome window as shown in Figure 7-2.

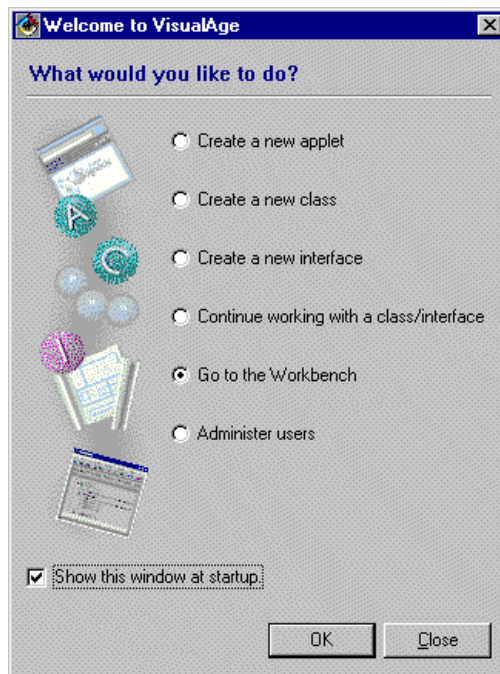


Figure 7-2 Welcome window for VisualAge for Java

2. Select **Go to the Workbench**. Click **OK**.
3. We now need to load some VisualAge features. Press F2. You are presented with the Quick Start window as shown in Figure 7-3.

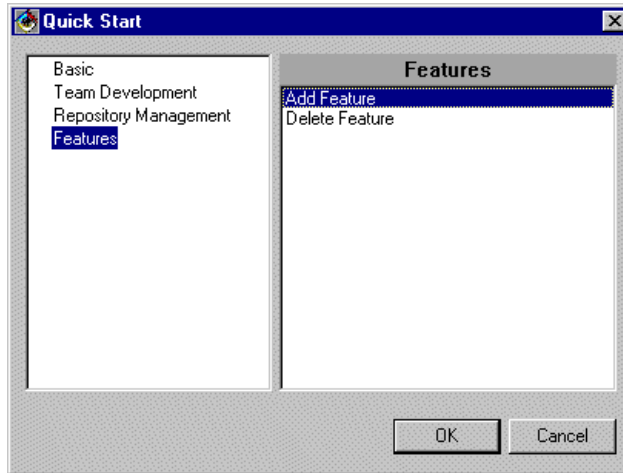


Figure 7-3 Quick Start window for VisualAge for Java

4. Select **Add Feature**, and click **OK**. A window appears listing all the available features for your installation of VisualAge.
5. Select the following features (holding down the Ctrl key will enable you to select multiple items):
  - **IBM EJB Development Environment 3.5.3**  
We use this to develop the Enterprise JavaBean in Section 7.4, “Using Enterprise JavaBeans” on page 217.
  - **IBM WebSphere Test Environment 3.5.3**  
This feature allows us to test the application from within VisualAge for Java. We use this throughout the development to test our application.
  - **Lotus Domino Java library 5.0.5**  
This feature provides support for Domino development within VisualAge for Java.

Figure 7-4 shows the features list as we completed it.

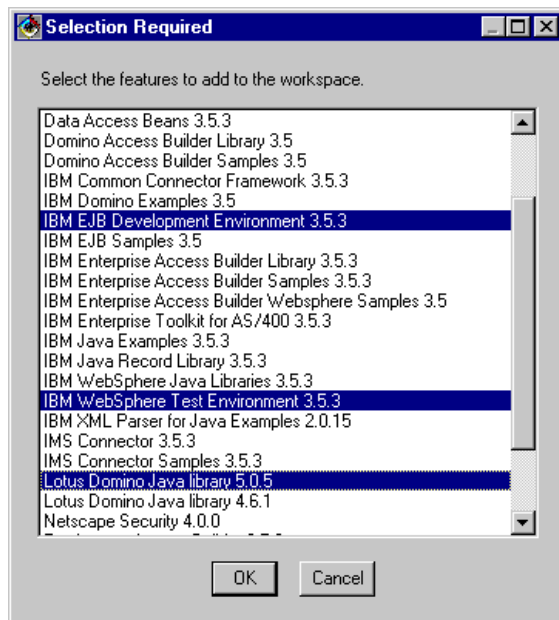


Figure 7-4 Features list for VisualAge for Java

6. Click **OK**. VisualAge loads the required features. At the completion of the process, the Workbench includes several extra projects and a new Workbench window labeled EJB. Figure 7-5 shows the Workbench.

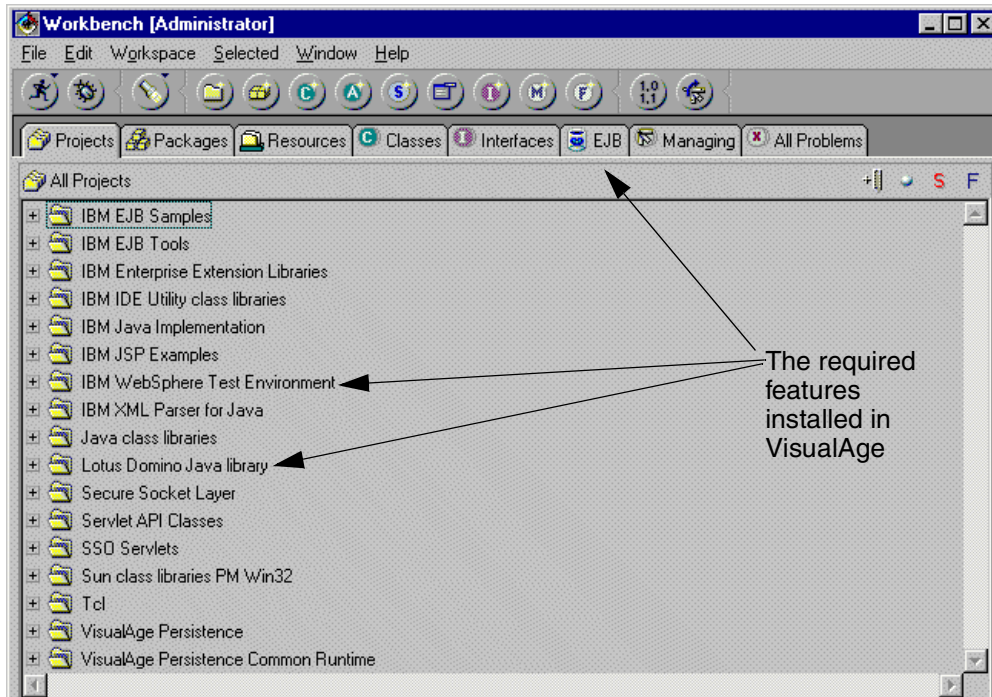


Figure 7-5 VisualAge Workbench with features loaded



## Creating a project

We now create a project. Projects are used as a logical grouping of packages.

1. From the toolbar, select the **New Project** icon as shown in Figure 7-6.

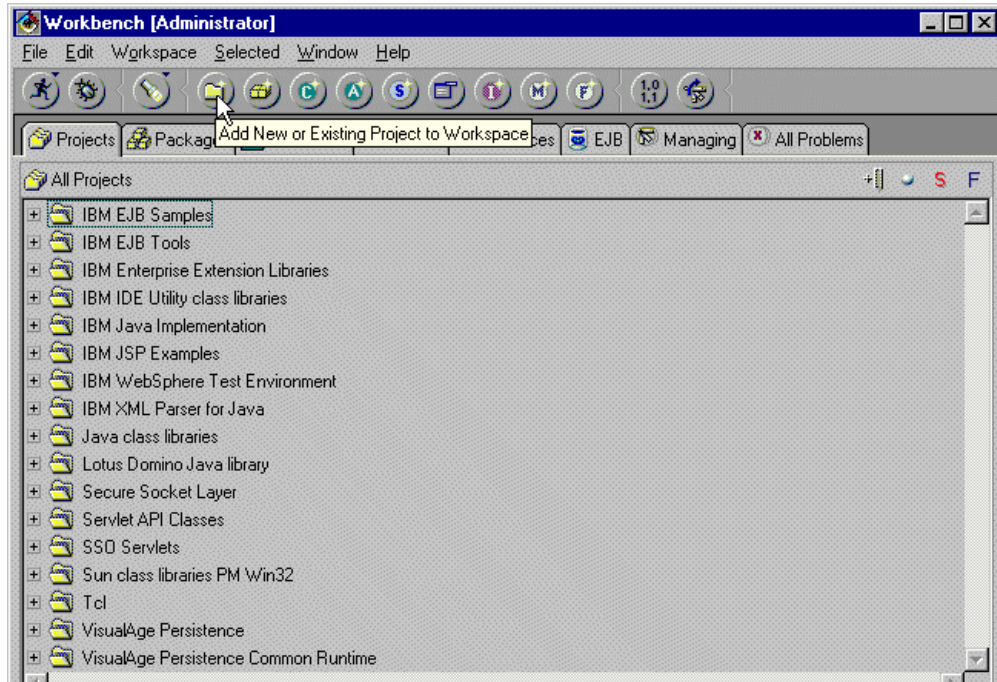


Figure 7-6 Adding a new project to the Workbench

2. This launches the new project SmartGuide. Create a new project called “Snow Dome Project”. The SmartGuide window as we filled it out is displayed in Figure 7-7. Click **Finish** to create the project.

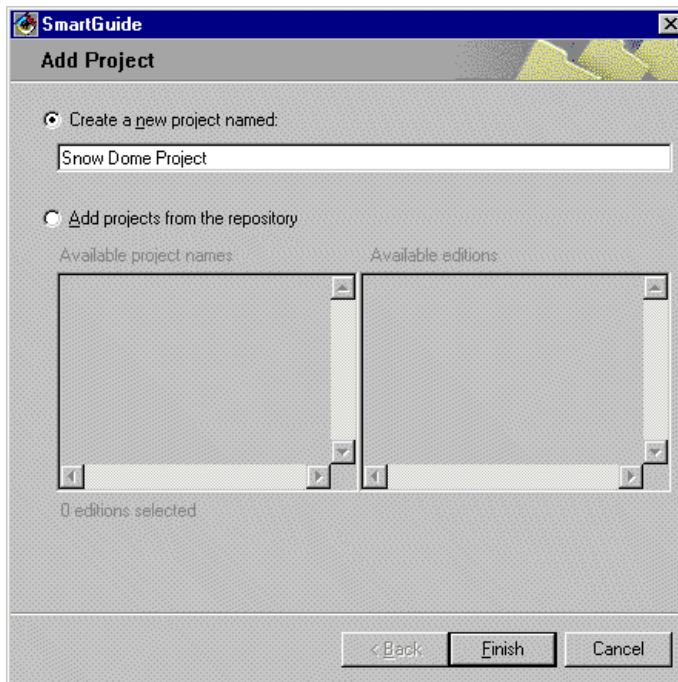


Figure 7-7 Defining a new project in VisualAge for Java

## Creating the packages

We now create a number of packages. Packages group class files into logical groups. We create two packages: one to store class files used by the EJB, and another to store the servlet class files.

1. From the Workbench, right-click the **Snow Dome Project** to display the context menu. Select **Add -> Package**. This will launch the Add Package SmartGuide.
2. Make sure the project name is Snow Dome Project. If not, click **Browse** and select it from the list.
3. The first new package name will be “snowdome.servlets”. Enter the information as displayed in Figure 7-8. Click **Finish**. A new package called “snowdome.servlets” is created.

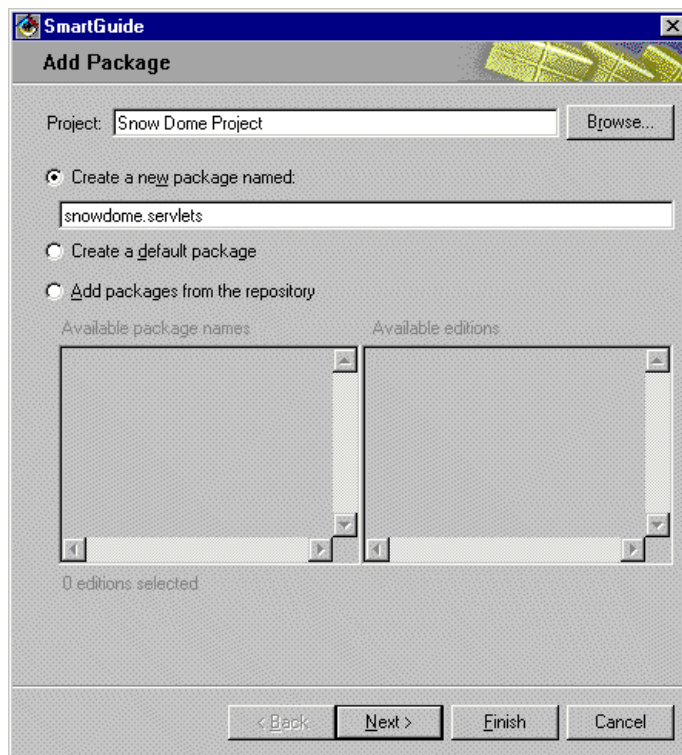


Figure 7-8 Creating a new package using the Add Package SmartGuide

4. Repeat the above steps to create another package called “snowdome.ejb”.

## Modify the default\_app for the WebSphere Test Environment

The final step in configuring VisualAge for Java is to modify the default\_app.webapp configuration file. This file is used by the test environment to locate servlets called during testing.

1. Using a text editor, open the default\_app.webapp file. This is found in the following directory.

```
X:\VAJAVA_INSTALL\ide\project_resources\IBM WebSphere Test
Environment\hosts\default_host\default_app\servlets
```

2. Add the following servlet entries to the file (see Figure 7-9).

**Important:** Make sure the entries are above the closing `</webapp>` tag at the end of the file. For a complete listing of this file, refer to “default\_app.webapp” on page 293.

```
<name>controller</name>
  <description>Controller Servlet for Snow Dome</description>
  <code>snowdome.servlets.Controller</code>
  <servlet-path>/webapp/sdWebApp/servlet/snowdome.servlets.Controller</servlet-path>
  <init-parameter>
    <name></name>
    <value></value>
  </init-parameter>
  <autostart>false</autostart>
</servlet>
<servlet>
  <name>register</name>
  <description>Servlet used for registration</description>
  <code>snowdome.servlets.Register</code>
  <servlet-path>/webapp/sdWebApp/servlet/snowdome.servlets.Register</servlet-path>
  <init-parameter>
    <name></name>
    <value></value>
  </init-parameter>
  <autostart>false</autostart>
</servlet>
<servlet>
  <name>registerwjsp</name>
  <description>Servlet used with JSP confirmation</description>
  <code>snowdome.servlets.RegisterWJSP</code>
<servlet-path>/webapp/sdWebApp/servlet/snowdome.servlets.RegisterWJSP</servlet-path>
  <init-parameter>
    <name></name>
    <value></value>
  </init-parameter>
  <autostart>false</autostart>
</servlet>
```

Figure 7-9 Modifying the default\_app.webapp file

This completes the basic configuration of VisualAge for Java.

### 7.1.3 Creating the Domino database

This section explains how to create the Domino database used in the Snow Dome application. The database is used to store registration information captured during the registration process. A database form is created to allow records to be displayed using the Domino client. A view is created to allow records to be searched for by the application components and Domino users.

Figure 7-1 on page 185 shows the registration database in the site architecture. All site activity leads to the Domino database. It acts as the data store for the application. Once the information is in the Domino database, it can be used as any other Domino data. We may choose to forward the documents to relevant people for approval. We could use the Domino mail services to notify external parties. Once we have captured the data, it is treated just as any other Domino data.

The basic steps involved in creating the example Domino database:

- ▶ Create a new Domino database called “registration.nsf”
- ▶ Create a form called “Registration”
- ▶ Create a view called “Attendees”

If you are familiar with Domino development and would prefer to download the complete database, refer to Appendix B, “Additional material” on page 295.

## Creating the new database

The conference registration database consists of a registration form and an attendees view. Create the Domino database using the following procedure.

1. Start your Lotus Domino Designer client.
2. From the menu, choose **File -> Database->New**. A new database window will appear.
3. In the Server field, type in the name of your Domino server or choose it from the pull-down menu.
4. For the title, type in Conference Registration.
5. For the file name, type registration.nsf.
6. For the template, choose **-blank-**, which is the default.
7. Select **Create full text index for searching**.
8. Click **OK** to create the database. Figure 7-10 shows the completed window.

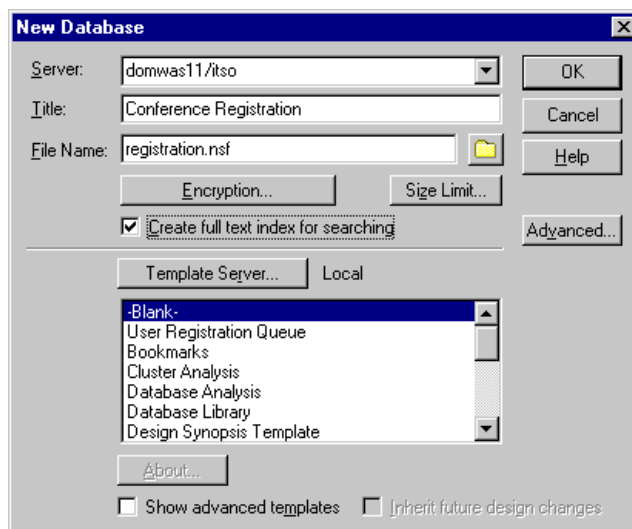


Figure 7-10 New database window

9. A pop-up window will ask if you would like to index the database now. Select **OK**.

## Creating the registration form

A Domino form is used to capture data to be stored in the database. The basic elements of a Domino form include:

- ▶ A form name
- ▶ Text labels
- ▶ Fields for data input
- ▶ Images

In this section we create the registration form as displayed in Figure 7-11.

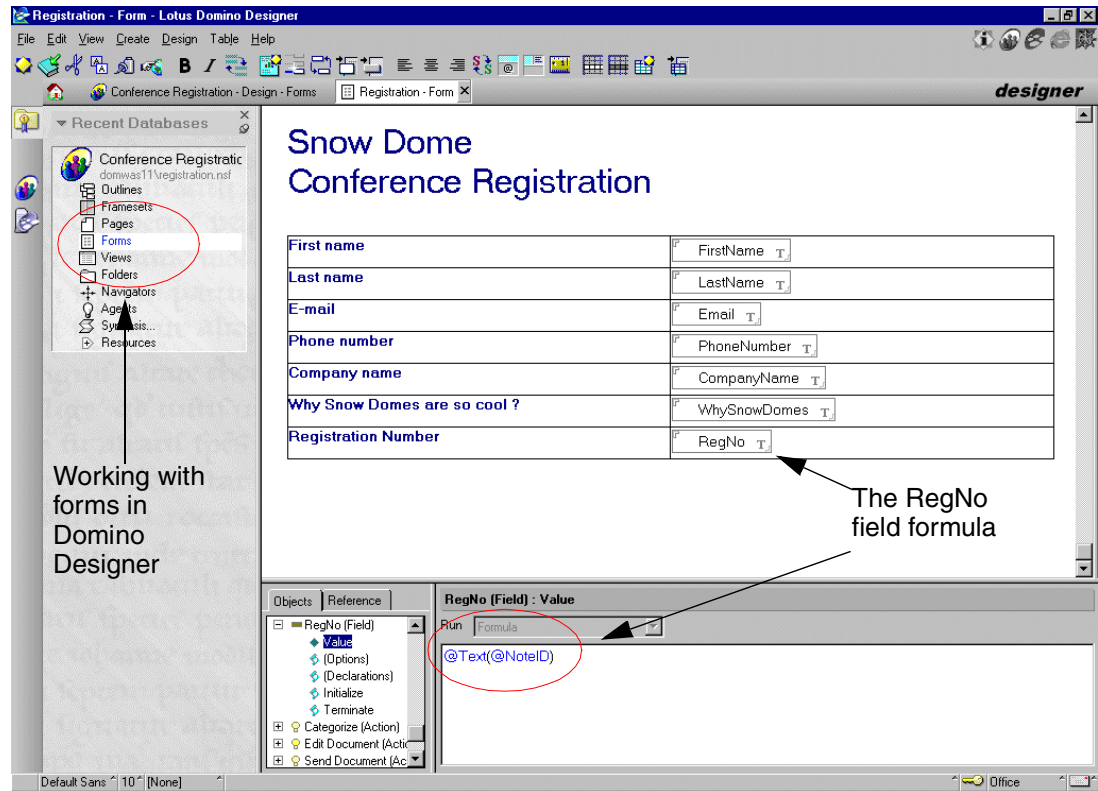


Figure 7-11 Registration form for the registration database

1. To create the form, select **Create -> Design -> Form**. A new blank form is displayed.
2. At the top of the form add the text used as the title for the form:  
Snow Dome Conference Registration
3. Below the text add a table. Select **Create -> Table**. The Create Table window appears.
4. For the number of rows, specify 7. For the number of columns, specify 2. Accept the default basic table type as shown in Figure 7-12 and click **OK**.

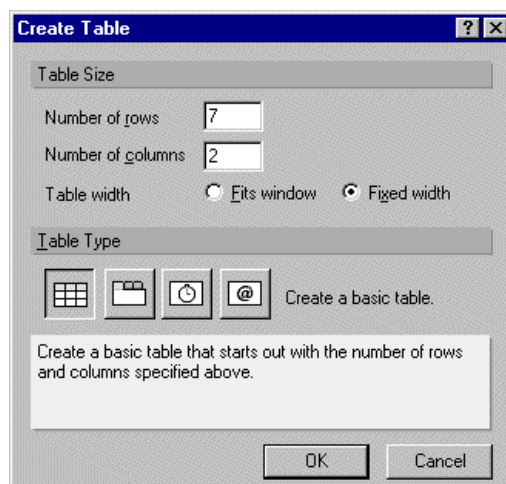


Figure 7-12 Create Table window

5. In the first cell of the left column type First name.
6. Move to the first cell of the right column and add a field by choosing **Create -> Field**.
7. In the Name field, type FirstName. Figure 7-13 shows the Field properties window.

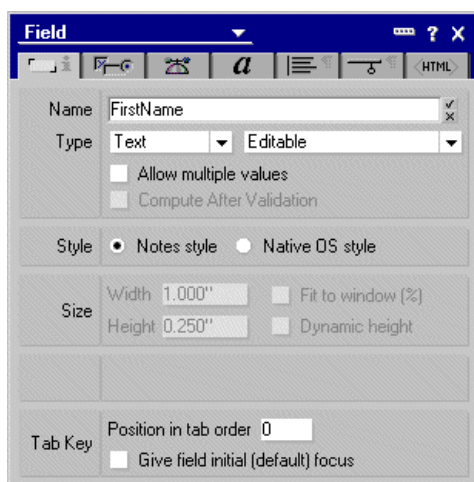


Figure 7-13 Properties window for a Domino field

8. Complete the table with the values shown in Table 7-2. Refer to Figure 7-11 as a guide for the layout of the form. Make sure field names are entered exactly as displayed. Any incorrect field names will lead to errors in later steps.

Table 7-2 Field information for Registration form

Label	Field Name	Field Type
Last name	LastName	Text, Editable
E-Mail	Email	Text, Editable
Phone number	PhoneNumber	Text, Editable
Company name	CompanyName	Text, Editable
Why Snow Domes are so cool?	WhySnowDomes	Text, Editable
Registration Number	RegNo	Text,Computed

Note that the last field, RegNo, is a computed field. We need to add a formula to this field. In the programmer's pane, enter the formula @Text(@NotelD) as shown in Figure 7-11.

9. Select **Design -> Form properties**. In the form properties box, type Registration as the name of the form.
10. Save the form and close it.

## Creating the attendees view

In order to display the list of attendees, we need to create a view. To create a view, perform the following steps:

1. Select **Create -> Design -> View**. The Create View window is displayed.
2. In the View Name field, type Attendees. Accept the default view type of Shared. Figure 7-14 illustrates the Create View window.

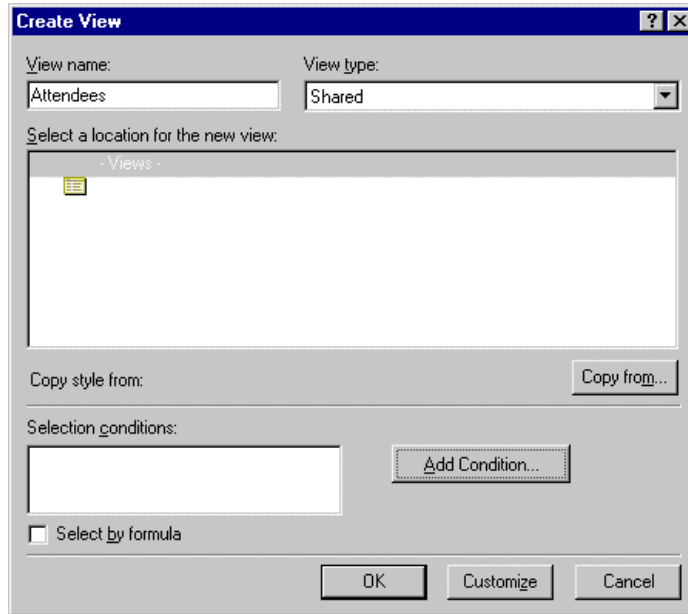


Figure 7-14 Create view window

3. Click the **Add Condition** button. This opens a window that allows you to specify rules for which documents are to appear in the view. In the Condition field, click the down arrow and select **By Form Used**.
4. Select **Registration** as shown in Figure 7-15. Click **OK**.

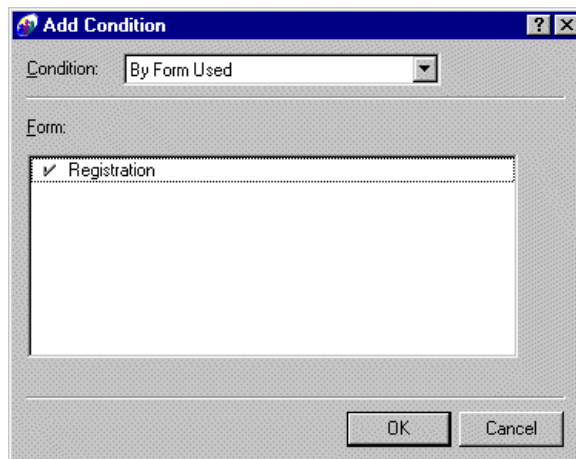


Figure 7-15 Specifying view selection criteria

5. On the Create View window, click **Customize** to open the newly created view in design mode.
6. Initially the view contains only one column with a title of #. Double-click the column header to display the column properties. Change the column title to Email.
7. In the programmer's pane, change the display type to Field. Select **Email** from the list of fields as shown in Figure 7-16.

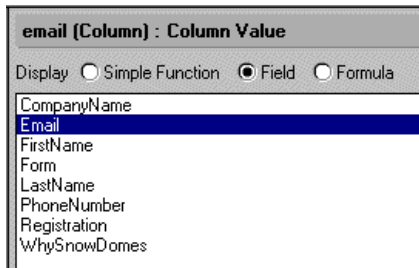


Figure 7-16 Specifying a column value

8. Select **Create -> Append New Column**. A new column is created. In the column properties specify the title of the column as Company.
9. Repeat step 7 to specify the column value. This time select the **CompanyName** field.
10. Repeat steps 7 and 8 to add columns for Firstname and Lastname.
11. Double-click the title of the Email column (which should be the first column on the left) to open the column properties box. Select the **Sorting** tab and choose **Ascending** as the sorting option. Figure 7-17 shows the sorting tab in the Column properties window.

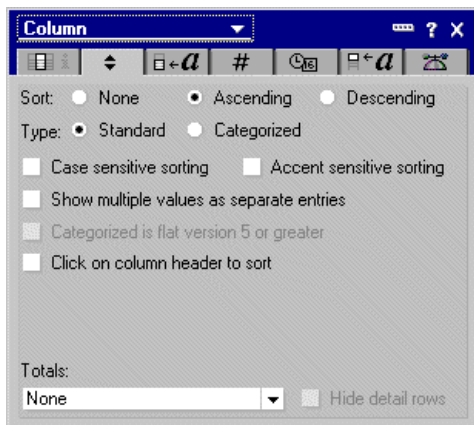


Figure 7-17 Setting column properties - Sorting tab

The completed view appears as shown in Figure 7-18.



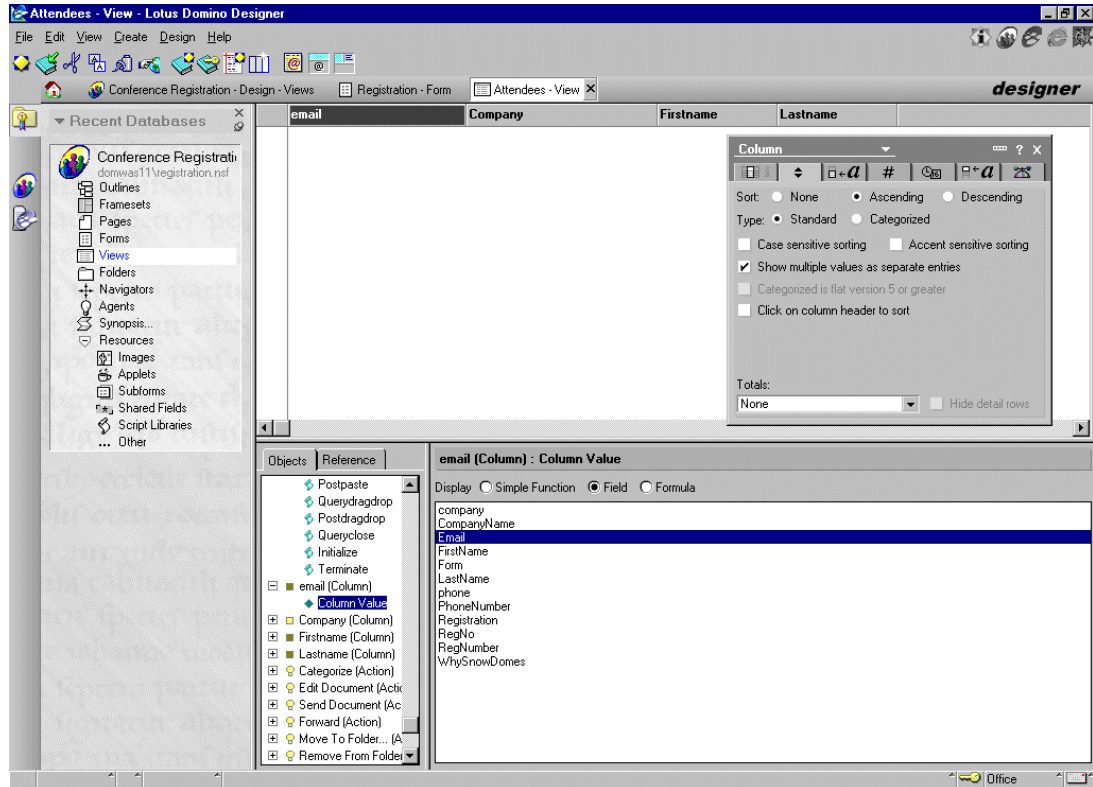


Figure 7-18 Attendees view

12. Save the view and close it.

This concludes the setup of the development environment.

## 7.2 Invoking servlets

We begin our sample application with a simple example of how you can use a servlet to process information gathered from a form. We will use this information to generate a registration document in the conference registration database.

For a detailed discussion on servlets and their life cycle, refer to the IBM Redbook *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, SG24-5755, found at: <http://redbooks.ibm.com/pubs/pdfs/redbooks/sg245755.pdf>

### 7.2.1 Servlet access to Domino

Our first example is a servlet that allows people to register for the Snow Dome World 2002 Conference. Figure 7-1 on page 185 shows the site architecture. In this topic we develop the register.class servlet. As can be seen from the diagram, four other files are needed for this example to function correctly.

- ▶ sdindex.html
- ▶ registration.html
- ▶ registration.nsf
- ▶ snowdome.gif

The sdsdindex.html file is the entry point for the application. We have a link from this file to registration.html. This is an HTML form. The action on this form is to call the Register servlet. The Register servlet uses the Notes remote classes to update the registration.nsf database.

The code for the two HTML files is included in Appendix A, “Code examples” on page 265. We created the registration database in Section 7.1.3, “Creating the Domino database” on page 191. The complete application is available for download. Refer to Appendix B, “Additional material” on page 295 for details.

Attendees will complete and submit a registration form. This will in turn call the Register servlet. The servlet reads the information provided in the form and creates a document in the registration database. Finally, the servlet generates a confirmation message and sends it back to the browser.

## Creating the Register servlet

If you have not already completed the setup of VisualAge for Java, refer to Section 7.1.2, “Setup of VisualAge for Java” on page 186, and in particular “Modify the default\_app for the WebSphere Test Environment” on page 190.

### *Creating the Register servlet stub code*

Stub code can be thought of as a template for a particular type of Java class file. In our example, VisualAge creates the stub code for the Register servlet. Once created, the Register servlet stub code will contain all the basic elements that should be implemented in a typical servlet. As an example, all servlets contain an init method. When creating the stub code for a servlet, the init method is included. VisualAge also creates a method called “performTask()”. This method is used as the entry point for all requests. In later sections, we customize the performTask method to control the flow of our servlets.

1. Start VisualAge for Java and select the **snowdome.servlets** package. From the context menu, select **Add -> Servlet** as shown in Figure 7-19.

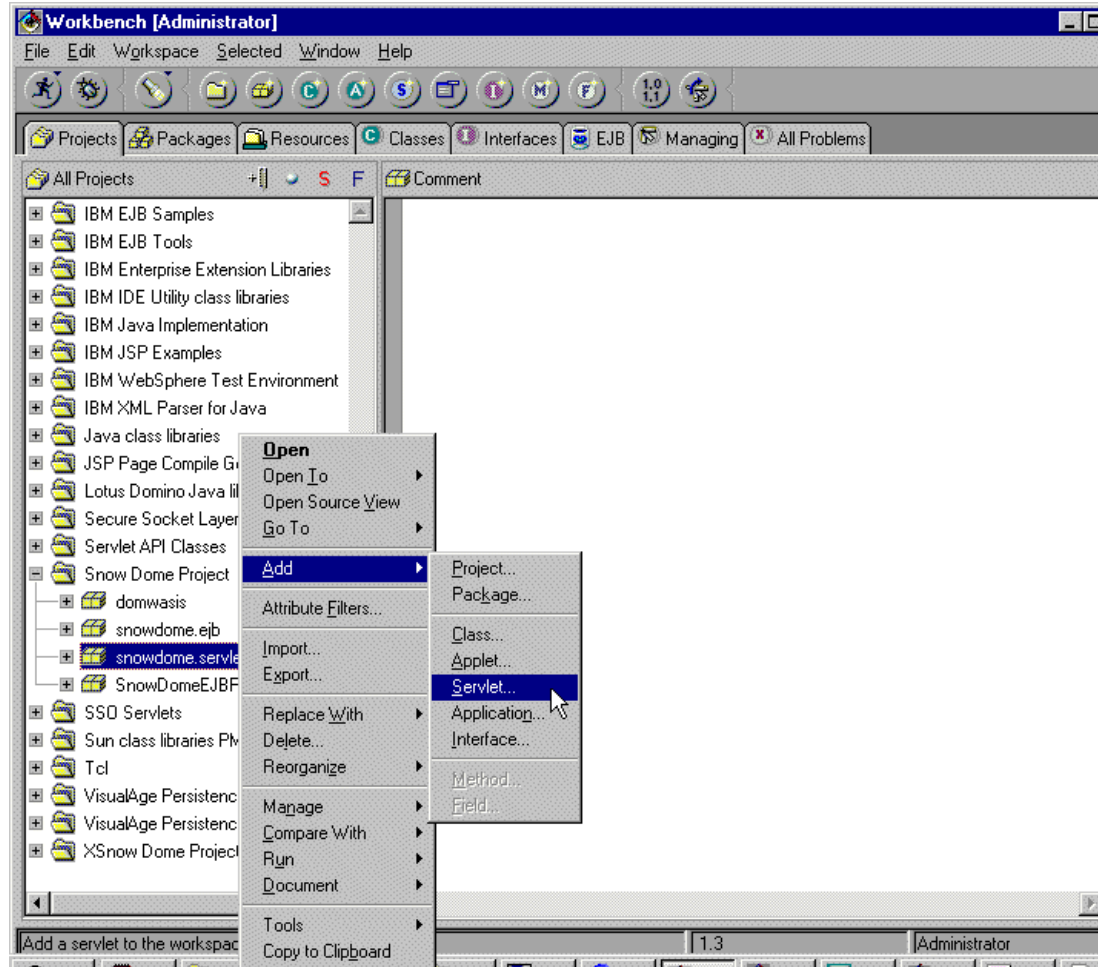


Figure 7-19 Creating the servlet

2. The Create Servlet SmartGuide is displayed. Complete the first window of the SmartGuide as shown in Figure 7-20 and then click **Next**.

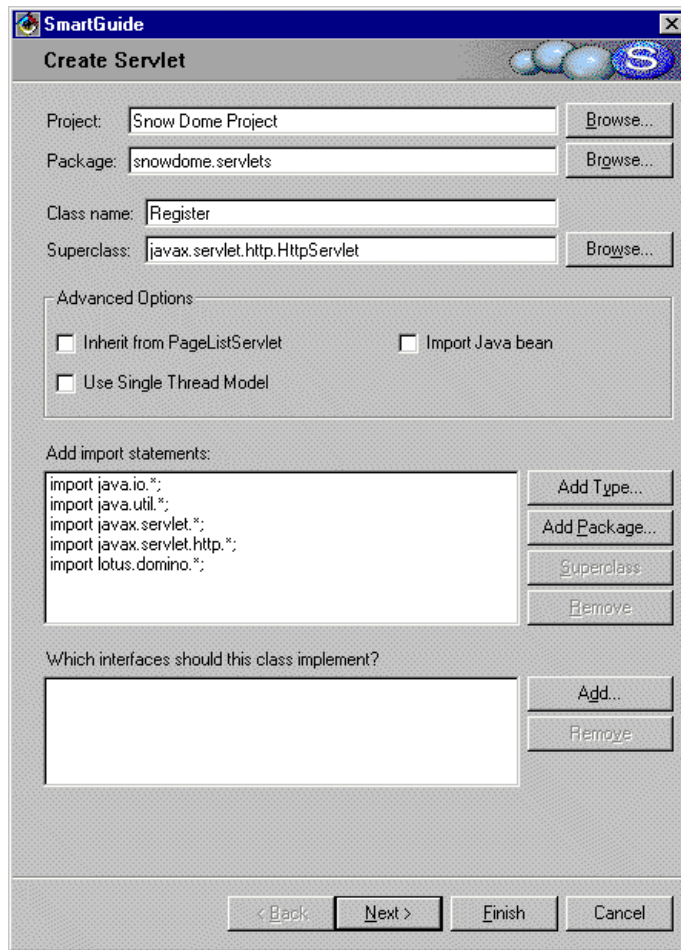


Figure 7-20 Create Servlet SmartGuide window 1

3. Complete the second window of the SmartGuide as shown in Figure 7-21. Click **Finish**. The SmartGuide will create the skeleton code of the Register servlet. We now need to add code, some fields, and some extra methods to give our servlet functionality.

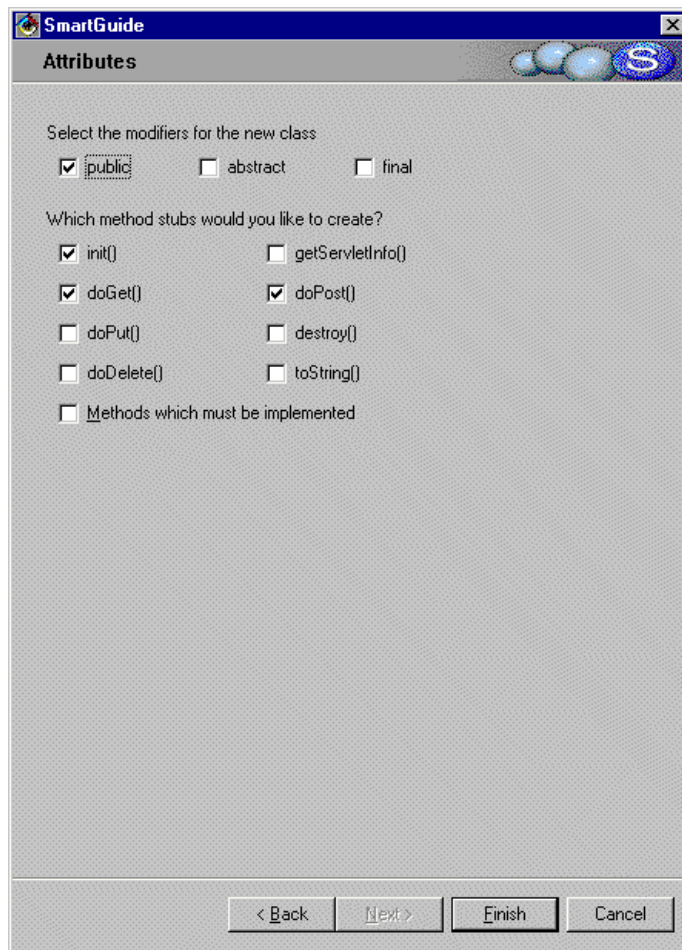


Figure 7-21 Create Servlet SmartGuide window 2

### Adding fields

Fields, or variables, allow data of a particular type to be stored in a class file. We use a number of fields to store information, such as the Domino session data and the Domino database. These fields are declared as instance variables of the Register servlet. All servlet instance variables are *available* to all instances of the servlet. By available we mean that the current value of the variable can be accessed by all instances of the servlet. This has many advantages, particularly when you need to share database connectivity. Be warned, you should think very carefully about creating instance variables that will be user specific. As an example, if you declared a field called “firstName” as an instance variable, the value of firstName may become corrupt when multiple instances try to set the value of this field.

Our example includes instance variables that are common to all servlet instances.

1. From the Workbench, select the **Register** servlet. The right-hand pane will display the class definition code.
2. Add the fields listed in Table 7-3 to the code as displayed in Figure 7-22.
3. Make sure you substitute the Domino server name and port with the correct values.

Table 7-3 Fields used in the Register servlet

Field name	Type
session	Session
database	Database
host	String
serverName	String
dbName	String
ior	String

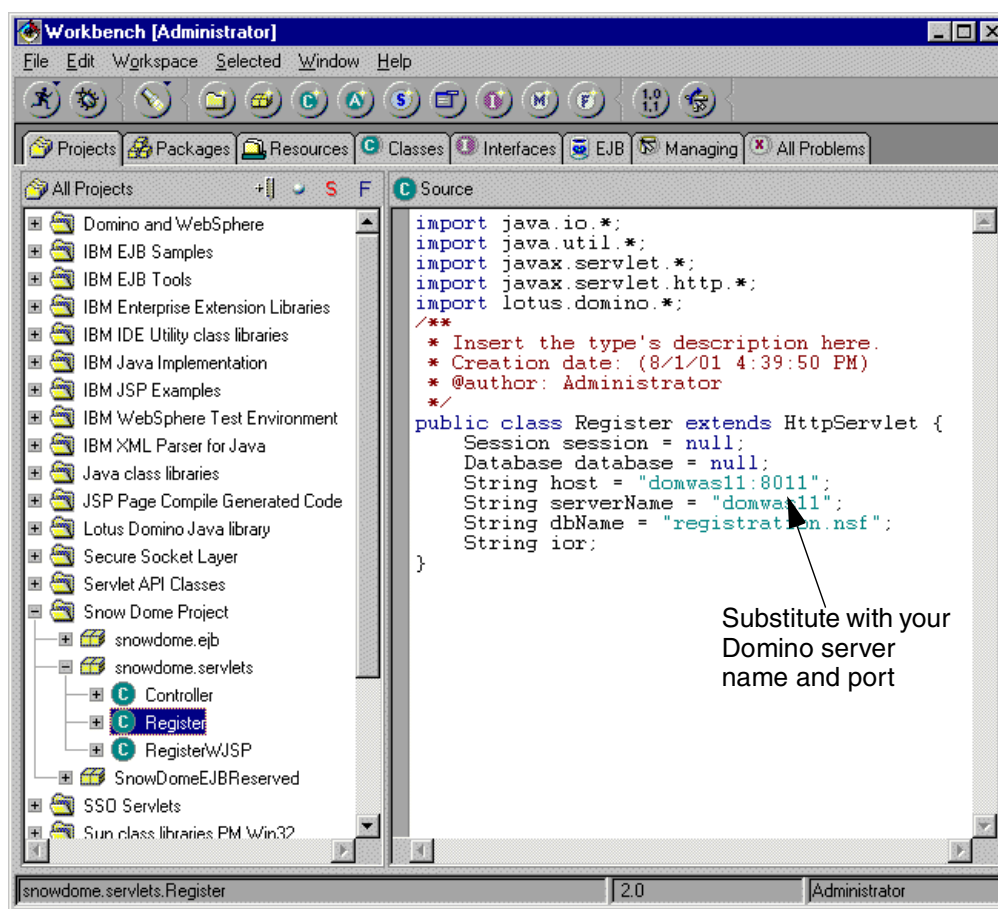


Figure 7-22 Adding fields to Register servlet

### Working with methods

The Register servlet requires two new methods. In this section, we create the createRegistration and writeConfirmMsg methods. We also modify the existing init and performTask methods.

1. From the Workbench, expand the Register servlet to display the existing methods. Select the **init** method and add the code as displayed in Figure 7-23. We use the init method to create a session and database objects. Since the init method is only called once (until the Servlet Engine is restarted), the session and database objects are only created once. Every instance of the servlet that is created has access to the session and database objects.

```

public void init() {
    if (session == null){
        try{
            ior = NotesFactory.getIOR(host);
            session=NotesFactory.createSessionWithIOR(ior);
            database = session.getDatabase(serverName,dbName);

        }
        catch (NotesException e){
            System.out.println("Error creating Notes session");
            e.printStackTrace();
        }
    }
}
}

```

Figure 7-23 init method code

2. Save your changes to the init method.
3. We now need to add the method createRegistration. This method is used to create the Domino document in the registration database.

Select the Register servlet and choose **Add -> Method** from the context menu, as seen in Figure 7-24.

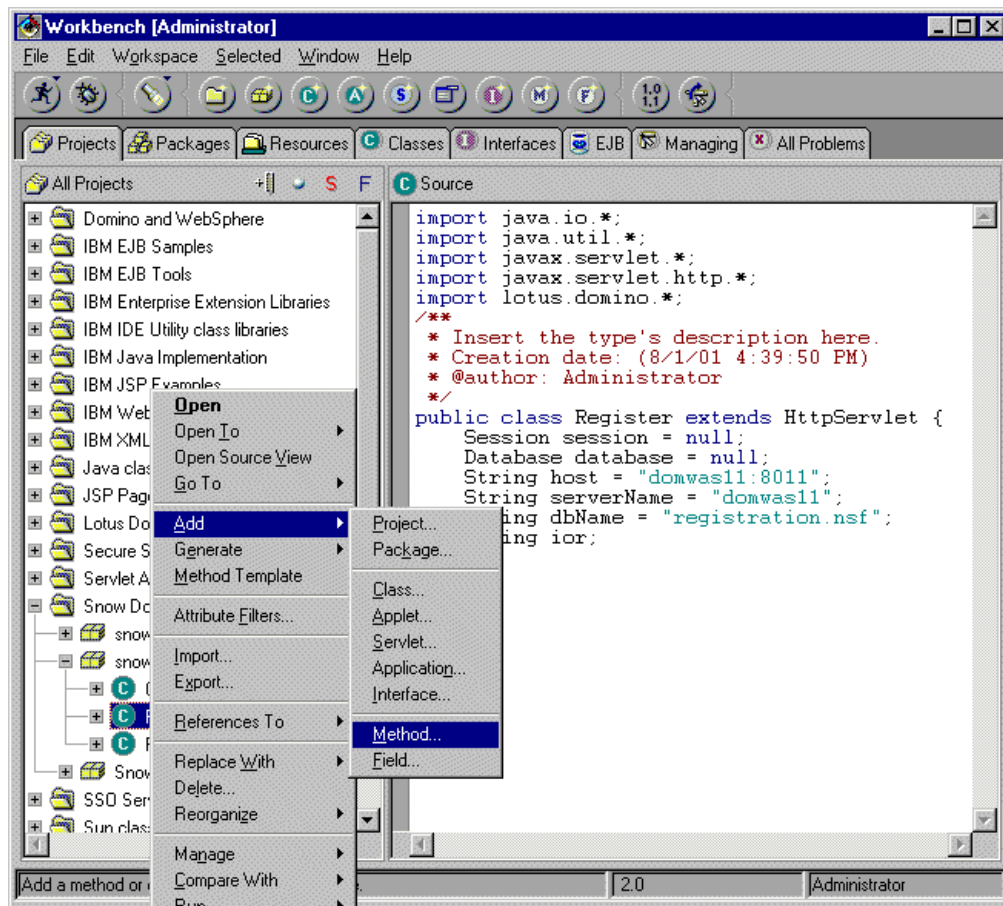


Figure 7-24 Add a method to the Register servlet



4. The Create Method SmartGuide is displayed. On the first window that is displayed, simply click **Next**.
5. Complete the second window of the SmartGuide, as seen in Figure 7-25. Click **Finish**.

Figure 7-25 Create Method SmartGuide

6. VisualAge creates a method signature for the createRegistration method. Complete the code as seen in Figure 7-26.

```
public void createRegistration(HttpServletRequest request) {
    try {
        //Create a document
        Document doc = database.createDocument();
        //Populate the document with the information gathered from the form
        doc.replaceItemValue("form", "Registration");
        doc.replaceItemValue("FirstName", request.getParameter("firstName"));
        doc.replaceItemValue("LastName", request.getParameter("lastName"));
        doc.replaceItemValue("PhoneNumber", request.getParameter("phone"));
        doc.replaceItemValue("CompanyName", request.getParameter("company"));
        doc.replaceItemValue("Email", request.getParameter("eMail"));
        doc.replaceItemValue("WhySnowDomes", request.getParameter("whySnowDomes"));

        //Save the Document and Generate Registration ID
        doc.save(true);
    }

    catch (NotesException e) {
        e.printStackTrace();
    }
}
```

Figure 7-26 The createRegistration method



7. We now need to create the `writeConfirmMsg` method. This method uses a `PrintWriter` object to send back a response to the browser. The method contains all the HTML code. It also uses some of the class fields from the request object to customize the response message.

Using the CreateMethod SmartGuide, create a new method called “`writeConfirmMsg`”. This method accepts two parameters. The first is called “out” and is of type `PrintWriter`. The second is called “request” and is of type `HttpServletRequest`. Figure 7-27 shows the completed SmartGuide. Click **Finish**.

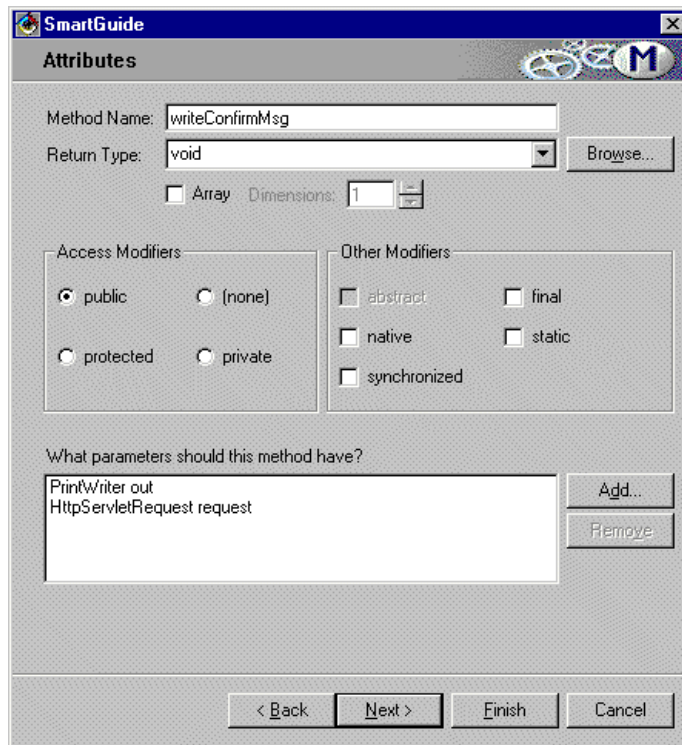


Figure 7-27 Create Method SmartGuide for the `writeConfirmMsg` method

8. Modify the `writeConfirmMsg` method with the code shown in Figure 7-28.

```
public void writeConfirmMsg(PrintWriter out, HttpServletRequest request){
    String firstName = request.getParameter("firstName");
    String lastName = request.getParameter("lastName");
    //Generate a confirmation message to be displayed to the user
    out.println("<!DOCTYPE HTML PUBLIC \"-//W3C// DTD HTML 4.0 Transitional
//EN\">");
    out.println("<html><head><title>Conference Registration
Complete</title></head><body>");
    out.println("<h1>Conference Registration Confirmation</h1><hr>");
    out.println("Thank-you, " + firstName + " " + lastName + " for registering for
the conference.<br>");
    out.println("We look forward to seeing you at Snow Dome World 2002");
    out.println("</body></html>");
}
```

Figure 7-28 `writeConfirmMsg` method

9. We now need to update the existing `performTask` method. The `performTask` method is the controller of the servlet. Its job is to simply call the relevant methods passing in the required objects. No actual work is performed by the `performTask` method.

Select the **performTask** method and update the code as displayed in Figure 7-29.

```
public void performTask(HttpServletRequest request, HttpServletResponse response) {

    PrintWriter out = null;

    try {
        response.setContentType("text/html");
        out = response.getWriter();
        createRegistration(request);
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    writeConfirmMsg(out, request);
}
```

Figure 7-29 *performTask* method for the Register servlet

This completes the development of the Register servlet. We now need to test that it is functioning as expected.

### Testing

VisualAge provides a complete testing environment we can use to test the Register servlet. This is the WebSphere Test Environment, which represents a lightweight version of the WebSphere Application Server. By using it, we are able to test and debug without having to deploy the servlet and HTML files to a server environment. For a complete discussion on VisualAge for Java and the WebSphere Test Environment, refer to *Programming with VisualAge for Java Version 3.5*, SG24-5264.

Before testing the Register servlet, ensure that you meet the following requirements:

- ▶ You have a copy of `registration.html` and `sdindex.html`.
- ▶ Your Domino server is running HTTP and IIOp server tasks.
- ▶ You have a copy of the registration Domino database (`registration.nsf`).
- ▶ You have modified your VisualAge for Java default\_app as shown in “Modify the default\_app for the WebSphere Test Environment” on page 190.

For details on downloading these files, refer to Appendix B, “Additional material” on page 295.

1. Copy `sdindex.html`, `registration.html`, and `snowdome.gif` files to the following directory:

```
X:\VAJAVA_INSTALL_DIRECTORY\ide\project_resources\IBM WebSphere Test
environment\hosts\default_host\default_app\web
```

2. Select **Workspace -> Tools -> WebSphere Test Environment**. This starts the WebSphere Test Environment. See Figure 7-30.
3. The WebSphere Test Environment Control Center is displayed (Figure 7-32 on page 208). From here we are able to configure, start, and stop various components of the test environment.

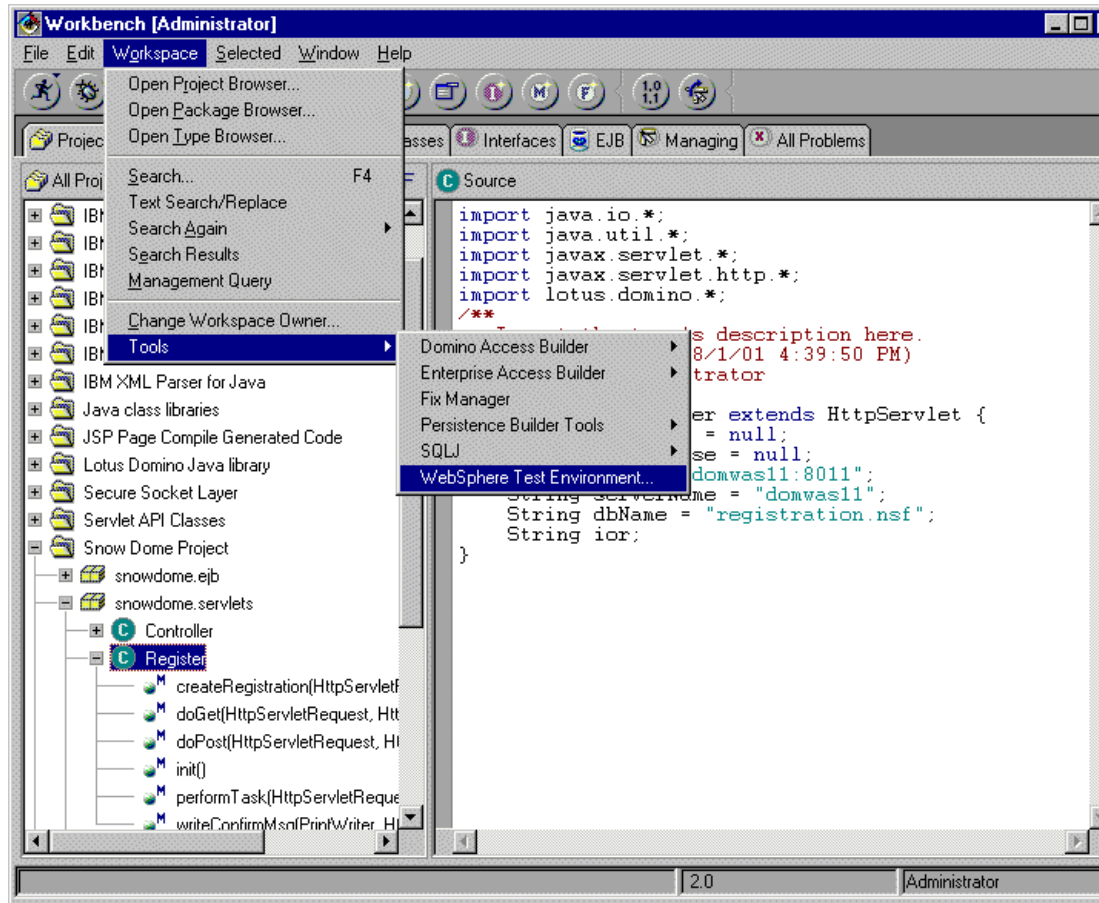


Figure 7-30 Starting WebSphere Test Environment

4. From the left pane of the WebSphere Test Environment Control Center, select the **Servlet Engine**. Click the **Edit Class Path** button. The Servlet Engine Class Path window is displayed (Figure 7-31). Click **Select All** and then **OK**.

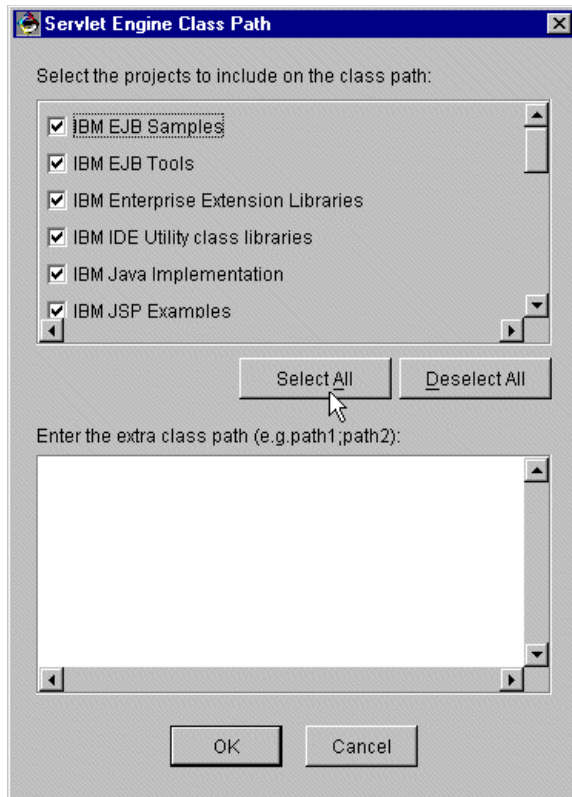


Figure 7-31 Servlet Engine Class Path window

5. From the WebSphere Test Environment Control Center, click **Start Servlet Engine**. When the Servlet Engine has started, the Control Center will look like Figure 7-32.

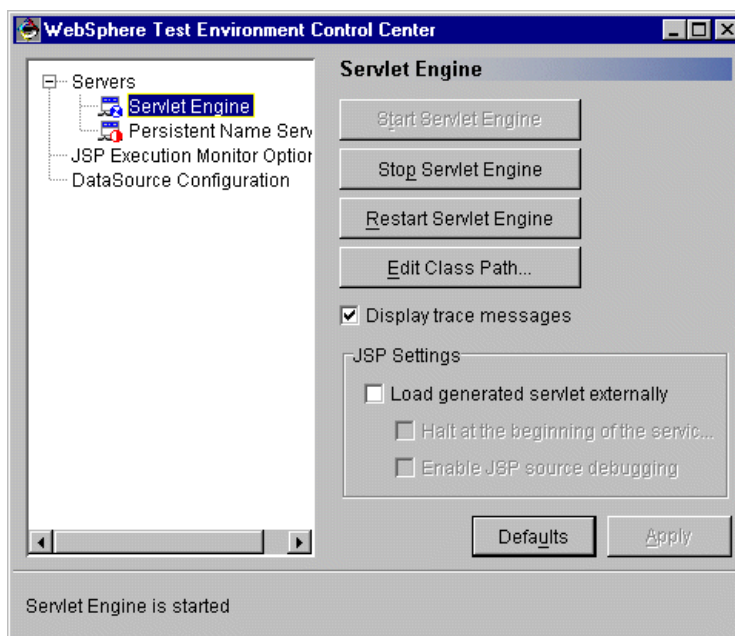


Figure 7-32 WebSphere Test Environment with Servlet Engine started

6. Open a Web browser and go to the URL:

`http://localhost:8080/sdindex.html`

Your browser will display the entry window to the Snow Dome application (Figure 7-33).

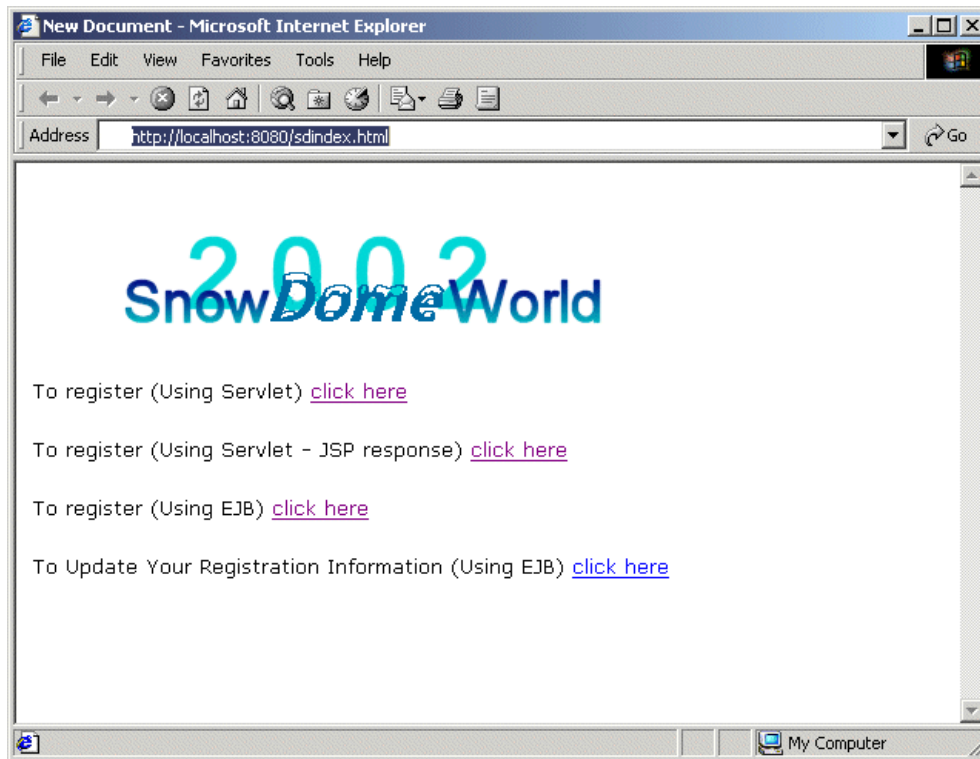


Figure 7-33 Snow Dome entry window

7. Select **To Register (Using Servlet) click here**. The registration form (registration.html) is displayed (Figure 7-34).

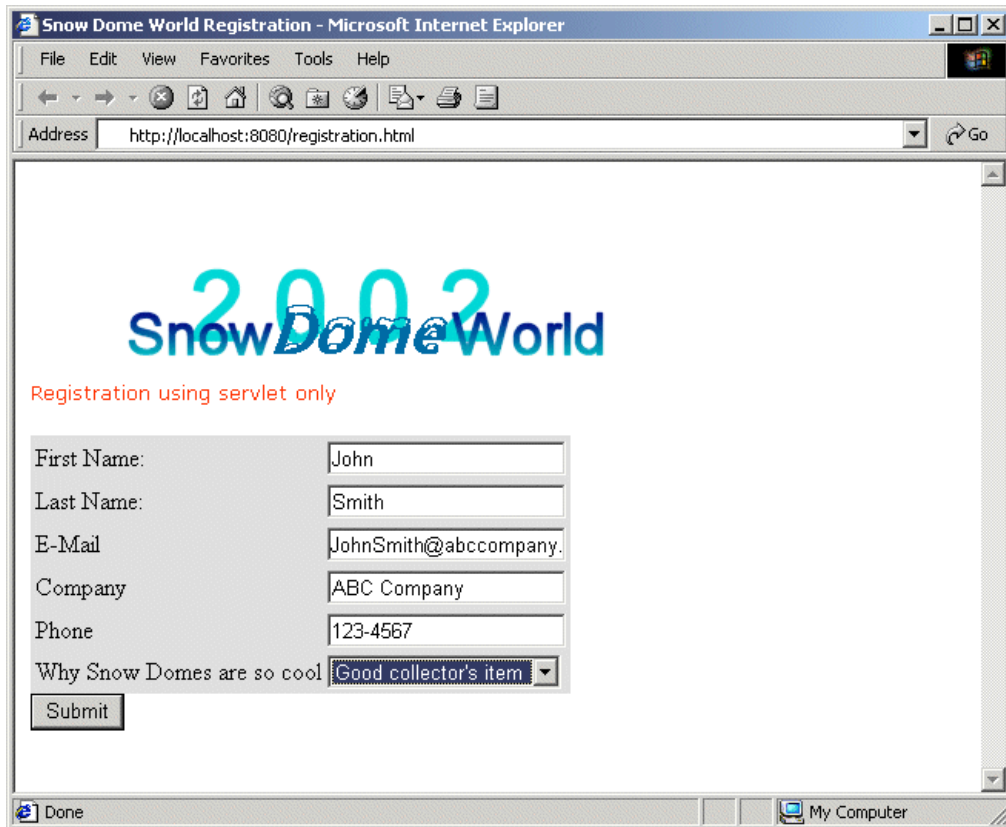


Figure 7-34 Registration.html

8. Complete the form with some sample data and click **Submit**. A record is created in the Domino database registration.nsf and a confirmation message is displayed to the user (Figure 7-35.).

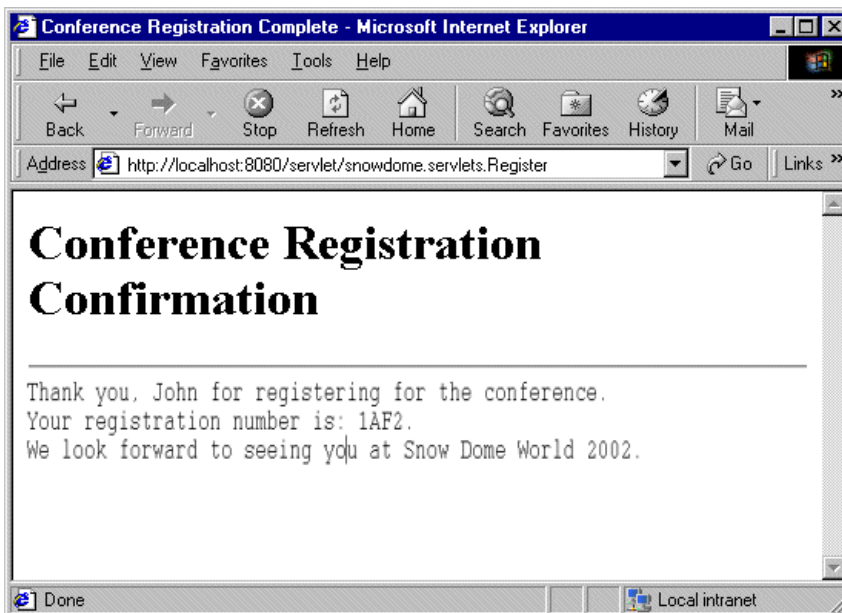
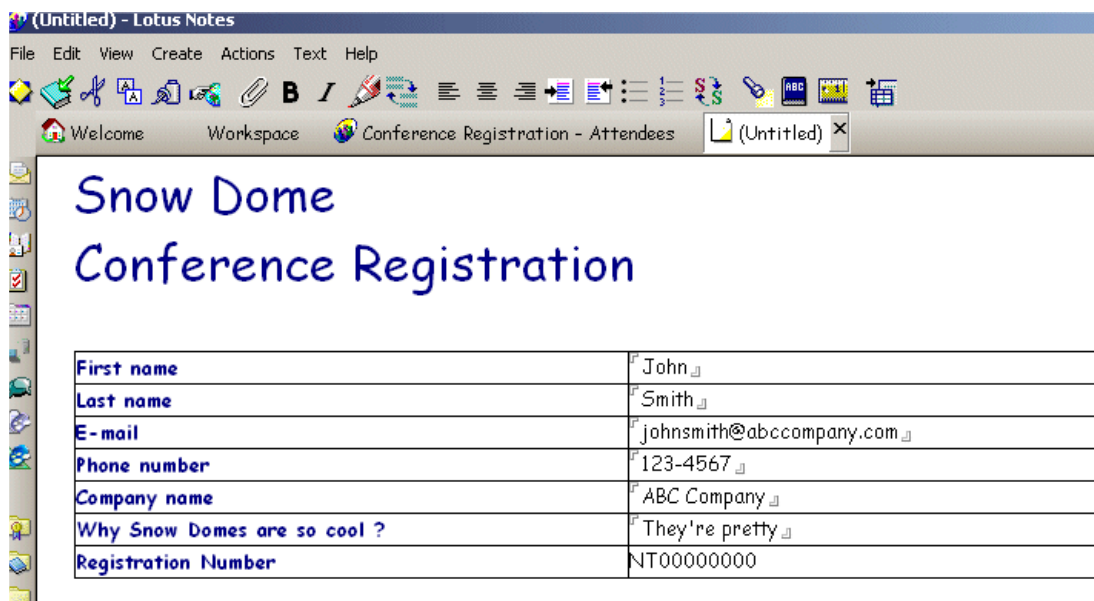


Figure 7-35 Confirmation window for registration

9. As a final step in testing, open the Domino database and look at the Domino record that was created (see Figure 7-36). Make sure the record was created as expected.



The screenshot shows the Lotus Notes application window titled '(Untitled) - Lotus Notes'. The menu bar includes File, Edit, View, Create, Actions, Text, and Help. The toolbar contains various icons for editing and navigation. The workspace area displays a document titled 'Snow Dome Conference Registration'. Below the title is a form with the following fields and values:

First name	John
Last name	Smith
E-mail	johnsmith@abccompany.com
Phone number	123-4567
Company name	ABC Company
Why Snow Domes are so cool ?	They're pretty
Registration Number	NT00000000

Figure 7-36 Registration record created in Domino

Congratulations! You have completed the first stage of development of the Snow Dome application. The next section discusses enhancing the servlet to use a JSP page to display the confirmation message.

## 7.3 Using JavaServer Pages

JavaServer Pages (JSPs) are HTML pages that contain Java code. JSPs provide a way of incorporating dynamic content into an HTML page. This is done by using *scripting elements*. These can be thought of as special HTML tags. The WebSphere server passes the file looking for these tags. When found, WebSphere takes action based on the type of tag or element found.

JSPs are converted to servlets by the WebSphere Application Server. This is usually done when the page is first called. You will notice that JSP pages take longer to display when first called, but run extremely fast on subsequent calls. It is possible to precompile a JSP before the page is called, but this is not necessary.

### 7.3.1 Extending the Snow Dome application using JSPs

In this section we extend the Snow Dome application to take advantage of JSP technology. Referring back to the site architecture (Figure 7-1 on page 185), we are now developing the RegisterWJSP servlet, and returnmessage.jsp. We also need a copy of the regwjsp.html file to complete the testing. All HTML files are included in Appendix A, "Code examples" on page 265.



## Creating the RegisterWJSP servlet

The RegisterWJSP servlet completes most of the same functionality that the Register servlet does. What does vary is the way that the response message is returned to the user. In the Register servlet a specific method (writeConfirmMsg) is responsible for generating the response message. The HTML is hard coded into the method. This approach has several disadvantages:

- ▶ The HTML code is imbedded in the servlet code.
- ▶ A Java developer is required to update any changes to the response message.
- ▶ Developing HTML this way is slow and inefficient.
- ▶ It is more difficult to take advantage of the many HTML authoring tools.

A far better approach is to use a servlet to do the processing, and a JSP to do the presenting. This approach offers:

- ▶ Java developers only need to code Java.
- ▶ HTML coders can create the presentation logic.
- ▶ Parallel development of look and feel and business logic is possible.
- ▶ Site maintenance is easy and efficient.

We begin by creating the new servlet logic. Rather than create a servlet from scratch, we will copy the existing Register servlet, and modify it to suit our needs.

1. From the Workbench window, select the **Register** servlet. Right-click and select **Reorganize -> Copy** (Figure 7-37).

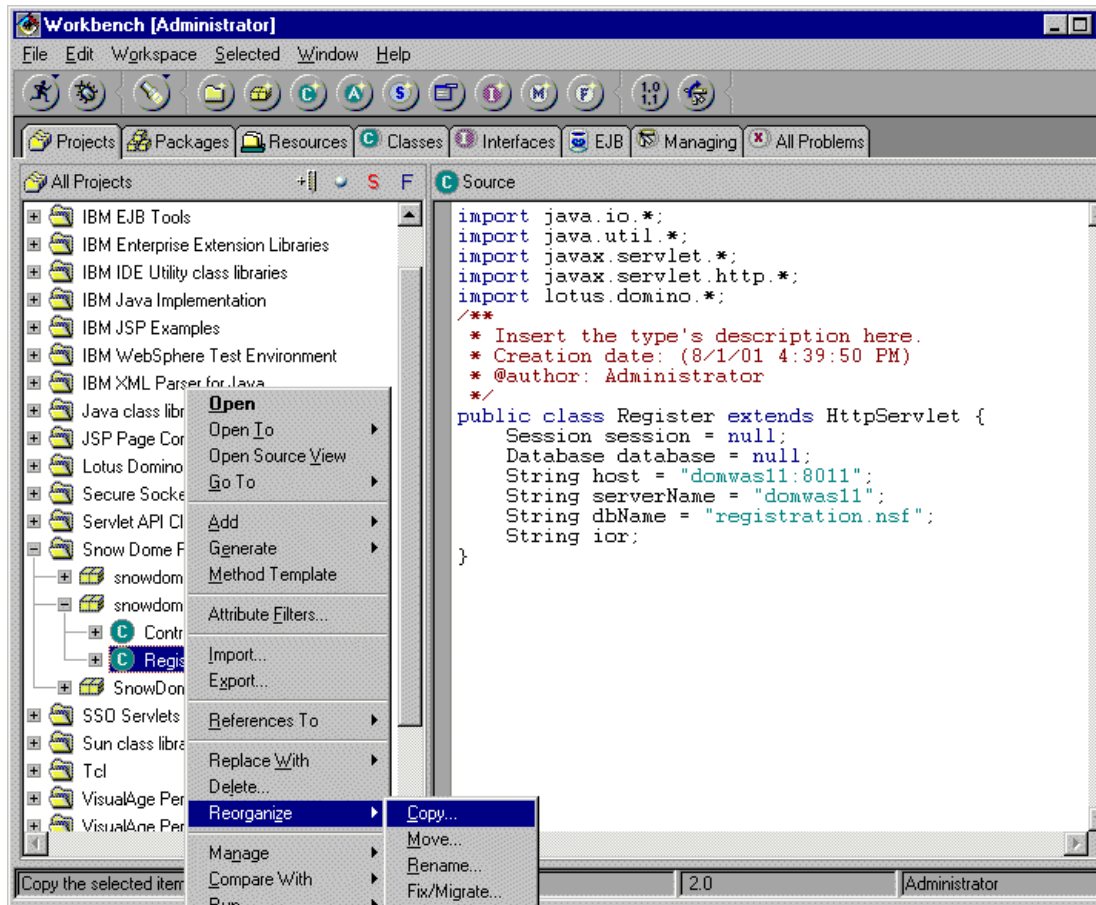


Figure 7-37 Copying a servlet



2. The Copying Types window is displayed. Make sure the snowdome.servlets package is selected and the **Rename the Copy** option is checked (Figure 7-38). Click **OK**.

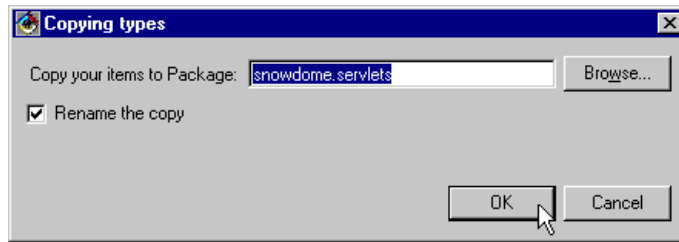


Figure 7-38 Copying Types window

3. Enter the new servlet name as RegisterWJSP and click **OK**. VisualAge creates the new servlet. We now need to modify the servlet to send the response to a JSP.



Figure 7-39 New servlet name

4. Expand the RegisterWJSP servlet and select the **writeConfirmMsg** method. Right-click and select **Delete**. This removes the method from our servlet. You will notice that an error is now highlighted in the performTask method. Ignore this for now.
5. Using the Create Method SmartGuide, add a method called “sendConfirmMsg”. This method accepts two parameters. Complete the method code as shown in Figure 7-40.

```
public void sendConfirmMsg(HttpServletRequest request, HttpServletResponse response){
    try{
        //Get the request dispatcher object
        String url="returnMessage.jsp";
        RequestDispatcher rd = getServletContext().getRequestDispatcher(url);

        //forward the request to returnMessage.jsp
        rd.forward(request,response);
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}
```

Figure 7-40 sendConfirmMsg code

This method is responsible for forwarding the request to the JSP. A RequestDispatcher is used to forward both the request and response objects to the JSP.

6. Update the performTask method as shown in Figure 7-41.

```
public void performTask(HttpServletRequest request, HttpServletResponse response) {  
  
    try {  
        createRegistration(request);  
    }  
    catch (Exception e) {  
        e.printStackTrace();  
    }  
    sendConfirmMsg(request, response);  
}
```

*Figure 7-41 performTask method for RegisterWJSP servlet*

We have made a number of changes. First, we no longer create a `PrintWriter` object. This is because the servlet is no longer responsible for sending information back to the client. Also, we no longer set a content type for the response object, for the same reason. The JSP is now responsible for presentation of the information. The final change is that we now use `sendConfirmMsg` rather than `writeConfirmMsg`.

This completes the creation of the `RegisterWJSP` servlet. We are now ready to create our JSP.

### **Creating returnMessage.jsp**

The `returnMessage` JSP is called by the `RegisterWJSP` servlet when it has completed updating the Domino database. The servlet passes two objects, the request and response. These can be used by the JSP to customize the response page. To create the JSP, perform the following steps:

1. In the `RegisterWJSP` servlet, we called a JSP named “`returnMessage.jsp`”. Using a text editor, create this file as listed in Figure 7-42.

```

<HTML>
<HEAD>
<%@page language="java" %>
<META name="GENERATOR" content="IBM WebSphere Page Designer V3.5 for Windows">
<META http-equiv="Content-Style-Type" content="text/css">
<TITLE>
Registration Confirmation
</TITLE>
</HEAD>

<BODY BGCOLOR="#FFFFFF">
<TABLE border="0">
  <TBODY>
    <TR>
      <TD><IMG src="/snowdomebanner.gif" width="500" height="100" border="0"></TD>
      <TD></TD>
    </TR>
  </TBODY>
</TABLE>
<P align="left"><FONT color="#000099" size="5" face="Arial">Registration Complete
</FONT><BR>
<BR>
<FONT face="Arial">Thank you <%=request.getParameter("firstName")%>
<%=request.getParameter("lastName")%> for registering for Snow Dome World 2001<BR>
<BR>
Your registration was complete on <%= new java.util.Date() %>.

<BR><BR>Return to <A HREF="/webapp/sdWebApp/SDIndex.html">home page</A>
</BODY>
</HTML>

```

Figure 7-42 returnMessage.jsp

## 2. Save the file to the WebSphere Test Environment Web directory:

X:\VAJAVA\_INSTALL\_DIRECTORY\ide\project\_resources\IBM WebSphere Test  
environment\hosts\default\_host\default\_app\web

**Important:** JSP files are case sensitive. When saving the file, make sure you save it as returnMessage.jsp.

This file is made up of mostly HTML tags with some JSP specific code. As an example:

```
<%=request.getParameter("firstName")%>
```

This line retrieves the value of the firstName parameter stored in the request object.

It is possible to store any Java object you like in the request object. As an example, you may like to store the complete registration details in a new Java object called “person”. The person object is then set in the request object and can be retrieved and used by the JSP.

For an excellent reference on JSPs and servlets, refer to *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, SG24-5755.

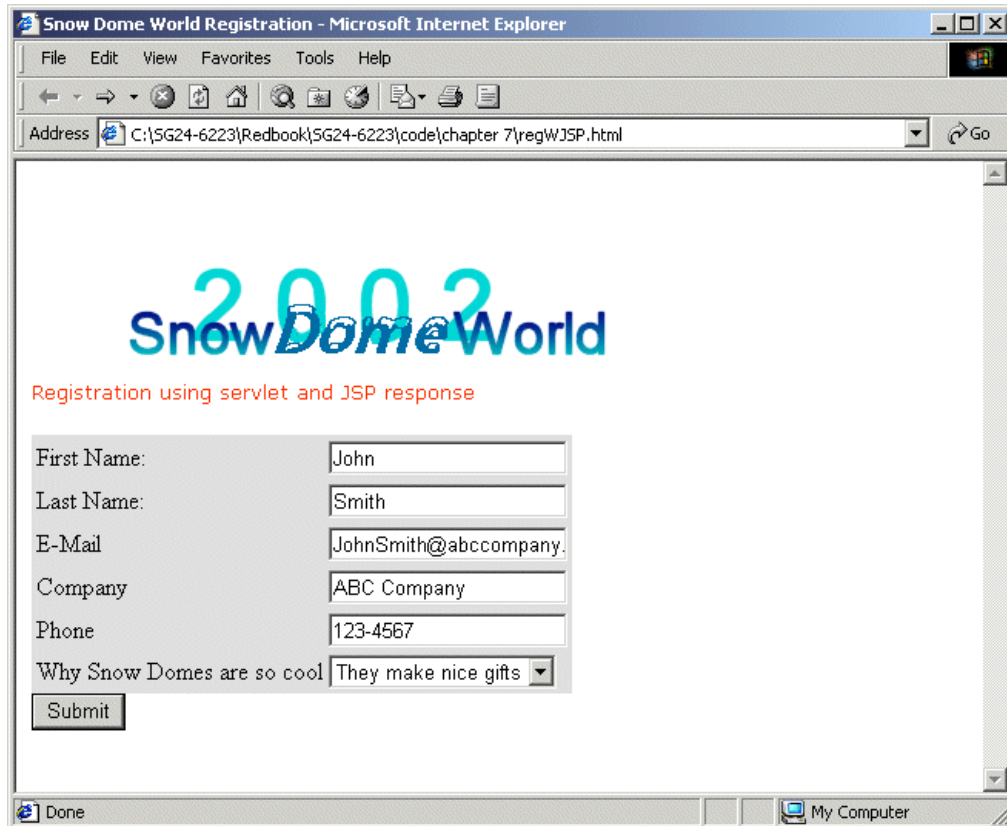
## Testing

Before testing, you will need to create or download the regWJSP.html file. Refer to Appendix A, “Code examples” on page 265, for a listing of the code.

1. Copy the file regWJSP.html to the VisualAge web directory.

X:\VAJAVA\_INSTALL\_DIRECTORY\ide\project\_resources\IBM WebSphere Test environment\hosts\default\_host\default\_app\web

2. Start the WebSphere Test Environment as described in “Testing” on page 206.
3. From the entry window of the application, select **To register (Using Servlet - JSP response)** [click here](#) (see Figure 7-33 on page 209).
4. Complete the registration form as shown in Figure 7-43, and click **Submit**.



The screenshot shows a Microsoft Internet Explorer window titled "Snow Dome World Registration - Microsoft Internet Explorer". The address bar displays "C:\SG24-6223\Redbook\SG24-6223\code\chapter 7\regWJSP.html". The main content area features the "Snow Dome World" logo in blue and green, with the text "Registration using servlet and JSP response" in red below it. A registration form is displayed with the following fields and values:

First Name:	John
Last Name:	Smith
E-Mail:	JohnSmith@abccompany.
Company:	ABC Company
Phone:	123-4567
Why Snow Domes are so cool:	They make nice gifts

A "Submit" button is located at the bottom left of the form area. The browser's status bar at the bottom shows "Done" and "My Computer".

Figure 7-43 Registering with a JSP confirmation

5. Confirm that the details in the confirmation window are correct (see Figure 7-44).

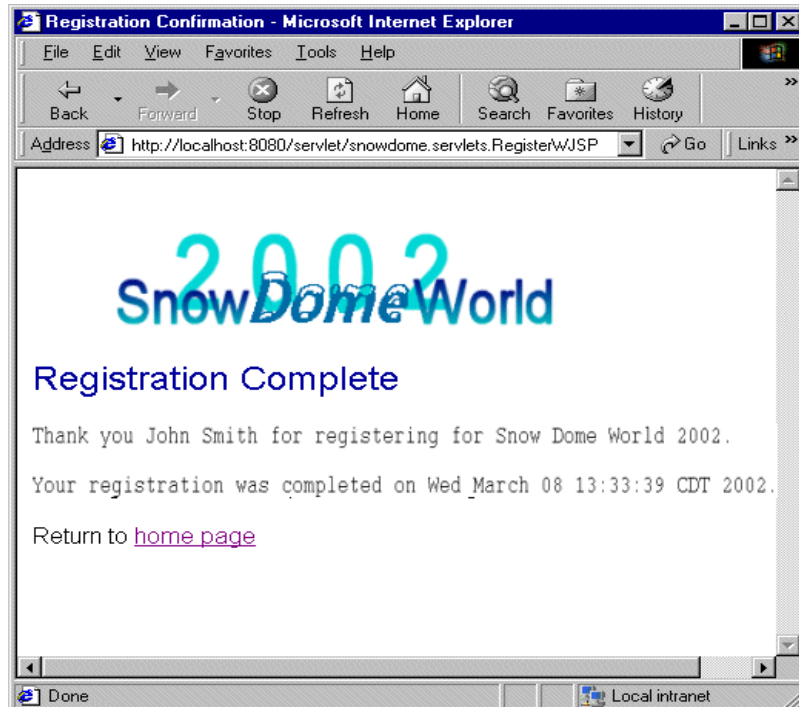


Figure 7-44 JSP returnMessage

6. Check the Domino database to ensure the document created as expected.

Our application now includes some JSP technology. In the next section we are going to add an EJB to complete the application.

## 7.4 Using Enterprise JavaBeans

Enterprise JavaBeans (EJBs) represent a powerful technology that enables developers to extend their applications to the enterprise. When and where you use EJBs depends vastly on the requirements. EJBs introduce an extra complexity into the development. In return they offer a high degree of flexibility. As a general rule EJBs are best suited to large and complex transactions.

We will now extend our Snow Dome example to show you how to take advantage of EJB technology.

People that have registered for “Snow Dome World 2002” need to be able to update their registration information. The developers have decided to use EJB technology to allow people to search for and update their details.

In the following sections, we create an entity bean with bean managed persistence (BMP) called “RegDoc”. We then create an AccessBean. A new servlet and a number of JavaServer Pages are introduced. Together these components will add the following new functionality:

- ▶ Search for registration details by e-mail address.
- ▶ Allow users to update existing details.
- ▶ Provide a confirmation page of changed details or new registrations.
- ▶ Provide error handling for records that cannot be located.
- ▶ Allow registration using EJB technology.

During development we test the application in VisualAge for Java. This environment offers a fully functioning test platform. Finally we deploy our application to the WebSphere Application Server.

To create the BMP bean you will need to have VisualAge for Java installed, and have loaded the required features. Refer to “Setup of VisualAge for Java” on page 186 for details on installing the features.

### 7.4.1 Using Enterprise JavaBeans to access Domino

EJBs can be grouped into two distinct types, Session beans and Entity beans. Each type can be further divided into another two sub-types. Session beans can be *stateful* or *stateless*. Entity beans can be CMPs (container managed persistence) or BMPs (bean managed persistence). A complete discussion of the differences between the two types is beyond the scope of this book. For an excellent source of EJB information, refer to *EJB Development with VisualAge for Java for WebSphere Application Server*, SG24-6144.

Our example uses a BMP bean. Each bean instance represents a registration record from the registration database. Before we begin creating the bean, we need to determine field mappings and any extra methods that may be required. Table 7-4 shows the field mappings between Domino, the HTML fields, and the BMP fields. The site architecture (Figure 7-1 on page 185 ) helps us determine what mappings are necessary. The diagram shows that the controller servlet receives requests from two forms (regejb.html and sdsearch.jsp). The controller servlet validates the information received, and calls the RegDoc EJB. The EJB is then responsible for updating the Domino database. A common confirmation page (sdsuccessupdate.jsp) is used to display the results to the user.

This is a common programming pattern. HTML and JSP forms are used to get user input, a servlet is used to manage authentication and process, and an EJB is used to manage data storage. If you are interested in learning more about design patterns, refer to *Design and Implement Servlets, JSPs, and EJBs for WebSphere Application Server*, SG24-5754.

Table 7-4 Domino to BMP field mappings

Domino Fields	HTML Fields	BMP Fields
FirstName	firstName	firstName
LastName	lastName	lastName
Email	eMail	eMail
CompanyName	company	company
PhoneNumber	phone	phone
WhySnowDomes	whySnowDomes	why
@NoteID	NOT USED	regNumber

We also need some extra methods in our bean:

- ▶ A method to manage connectivity via IIOP between WebSphere and Domino.
- ▶ A method to search for records using a member's e-mail address.
- ▶ Setter and getter methods for all field values.

In the next few sections, we will create the RegDoc BMP bean using the above as a guide.

## Creating the EJB Group

This section will explain in detail how to create an enterprise bean group. Groups are used as a logical store for a group of related EJBs. The group may include both entity beans and session beans.

1. From the Workbench, select the **EJB** tab to display the EJB window (Figure 7-45). We will use the EJB window for all EJB development.

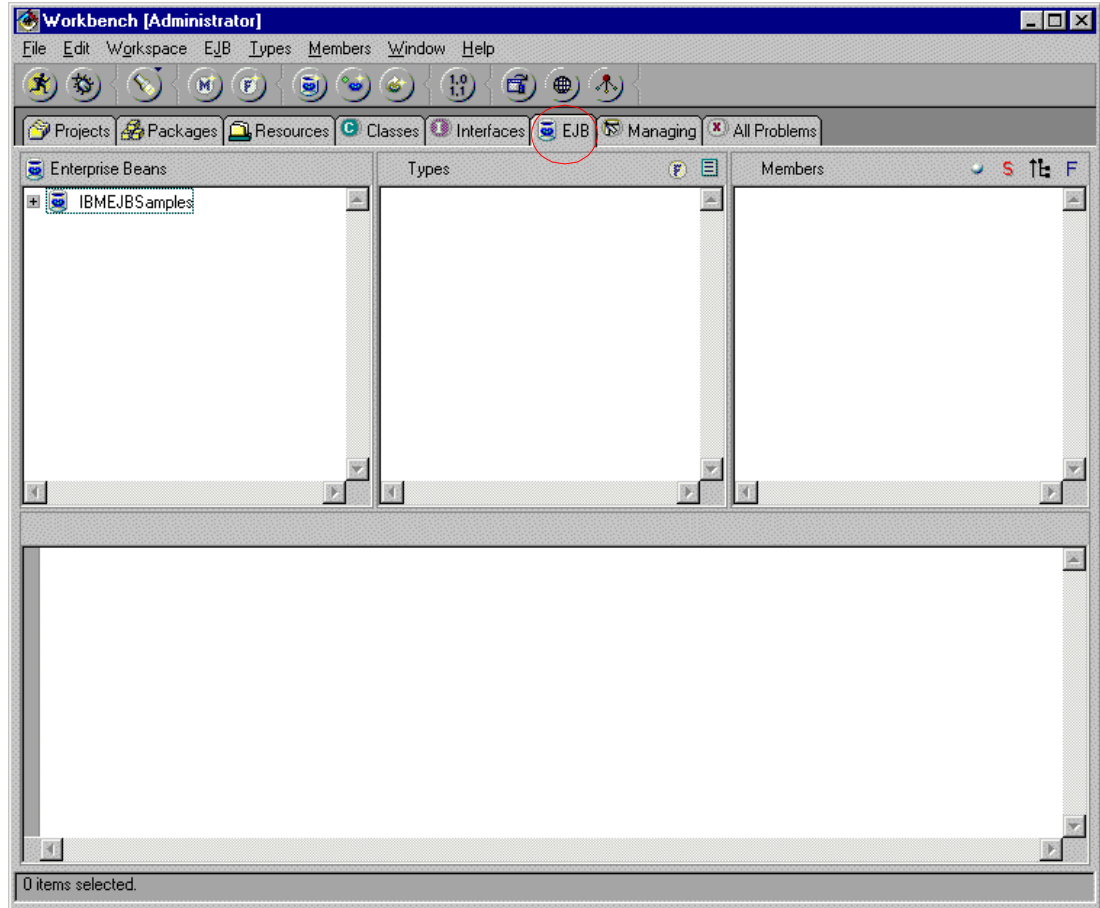


Figure 7-45 VisualAge EJB window

2. Select **EJB -> Add-> EJB Group** to display the Add EJB Group SmartGuide. Complete the SmartGuide fields as shown in Table 7-5. Figure 7-46 shows the completed SmartGuide. Click **Finish**.

Table 7-5 Field values for Add EJB Group SmartGuide

Field	Value
Project	Snow Dome Project
Create a new EJB group named	SnowDome

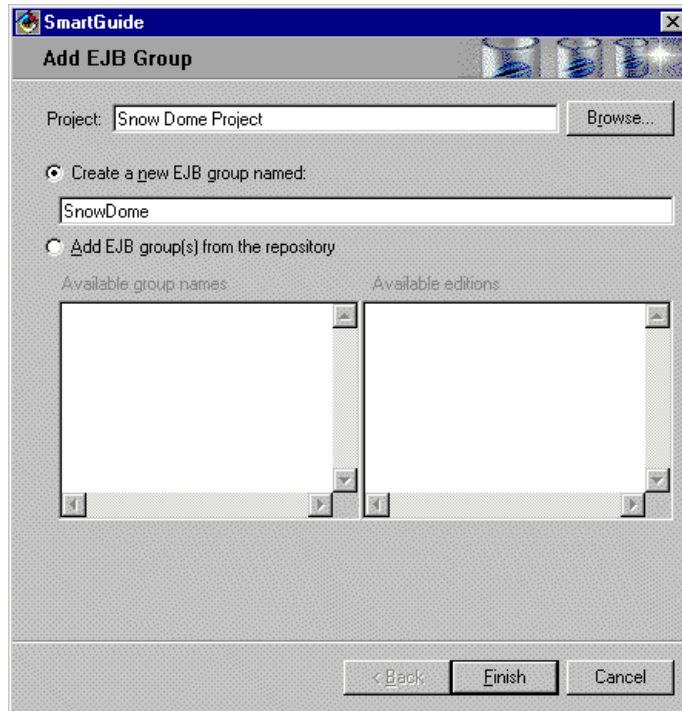


Figure 7-46 Add EJB Group SmartGuide

Your EJB window now includes a group called “SnowDome”. Figure 7-47 shows the new group as displayed on the EJB window. The next step is to add the enterprise bean.



Figure 7-47 EJB window with SnowDome group

### Creating a BMP enterprise bean

This section explains how to add the RegDoc bean to the SnowDome EJB group. The bean will be an entity bean and field persistence will be managed by the bean itself. Therefore we are creating a BMP bean.



1. Right-click the **SnowDome** EJB group to display the context menu. Select **Add -> Enterprise Bean**. This displays the Create Enterprise Bean SmartGuide. Complete the SmartGuide as shown in Table 7-6.

Table 7-6 Field values for Create Enterprise Bean SmartGuide

Field	Value
Create a new enterprise bean	Selected
Bean name	RegDoc
Bean type	Entity bean with bean-managed persistence (BMP) fields
Create a new bean class	Selected
Project	Snow Dome Project
Package	snowdome.ejb
Class	RegDocBean

Figure 7-48 shows the completed SmartGuide.

Figure 7-48 Adding the RegDoc Enterprise Bean

2. Click **Next** to display the Define Bean Class Attributes and Interfaces window.
3. Leave all values as their default, and click **Add Package**.

4. From the search list select the **lotus.domino.\*** package. Figure 7-49 shows the Import Statement window.

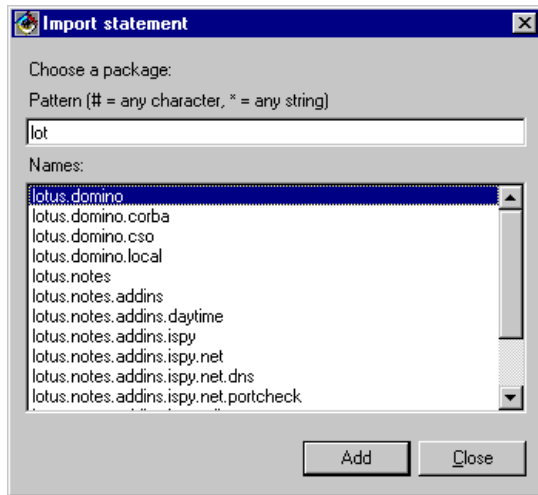


Figure 7-49 Importing the domino package

5. When you have finished adding the package, the SmartGuide window should appear as shown in Figure 7-50. Click **Finish** to create the bean.

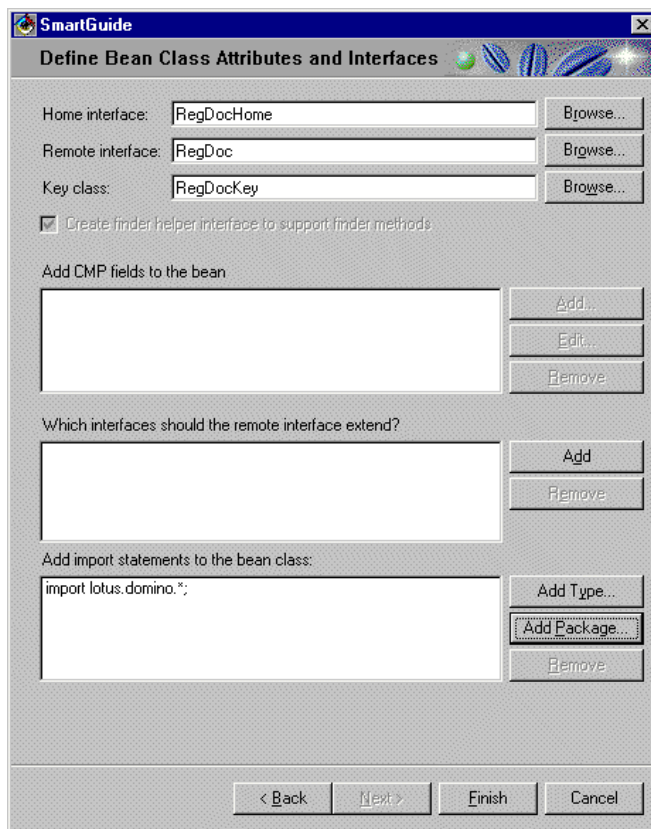


Figure 7-50 Import the lotus.domino package

You have now created the RegDoc bean. Your EJB Workbench window should look like Figure 7-51.

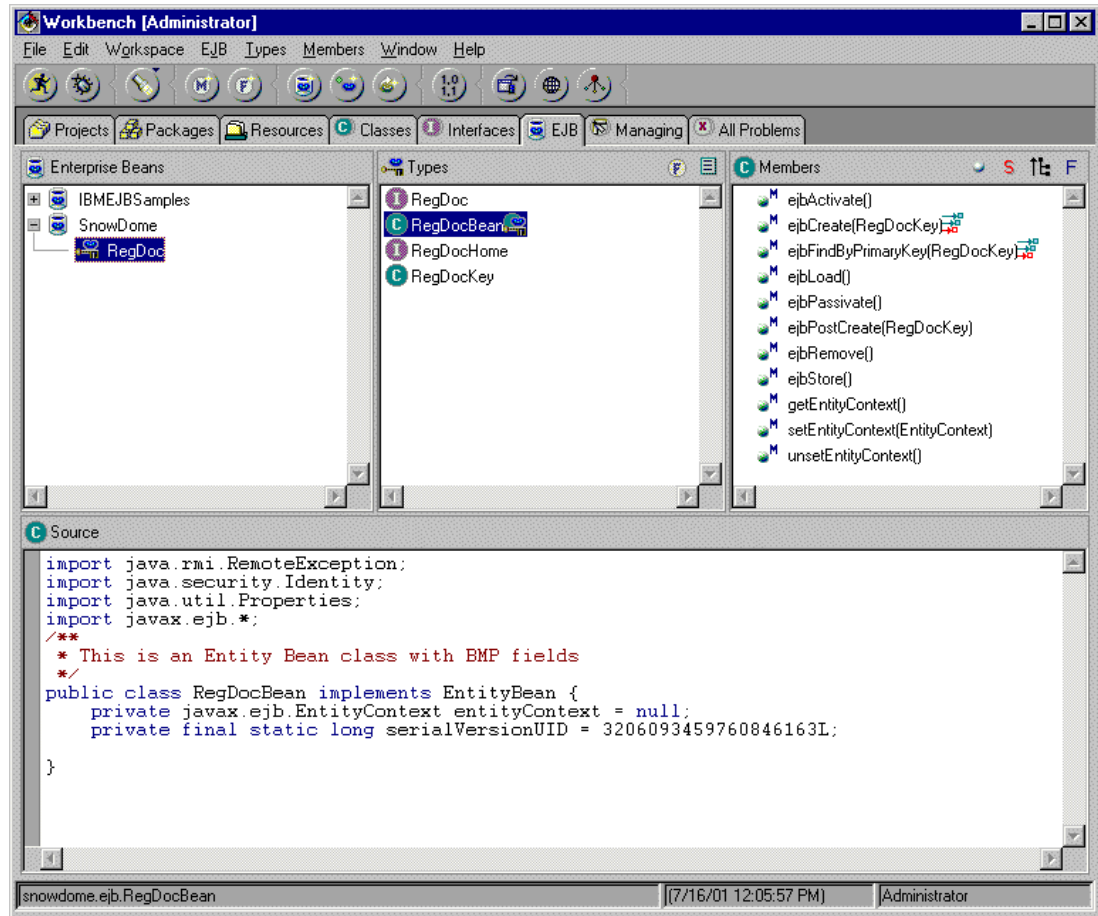


Figure 7-51 EJB Workbench with the RegDoc bean

## Adding fields

This section explains how to add the bean-specific fields to the RegDoc bean. Two sets of fields will be created. The first group represents the registration data as stored in Domino, namely:

- ▶ firstName
- ▶ lastName
- ▶ eMail
- ▶ phone
- ▶ company
- ▶ why
- ▶ regNumber

All these fields are declared private, and all have public getters and setters methods. The exception to this is regNumber, which is the key for each Domino record. This field will have a private setter method.

We also need to create four fields to manage connectivity to the Domino server.

- ▶ session
- ▶ database
- ▶ view
- ▶ document

These fields will be declared as private static fields. The exception to this is the document field, which will be declared private only. None of these fields will have setters or getters.

There are two methods for creating fields. The first method involves using the VisualAge SmartGuide. This is useful when creating fields that will have getter and setter methods. We use the SmartGuide to create the first group of fields.

The second method involves adding the field definitions directly into the bean source code. For fields that do not require setter and getter methods, this method is much faster. We use this method to create the fields involved in the Domino server connectivity.

To create a field using the field SmartGuide:

1. Select **RegDocBean** from the Types pane of the EJB window. From the EJB toolbar, select the **Create Field** icon (Figure 7-52). This will display the Create Field SmartGuide.

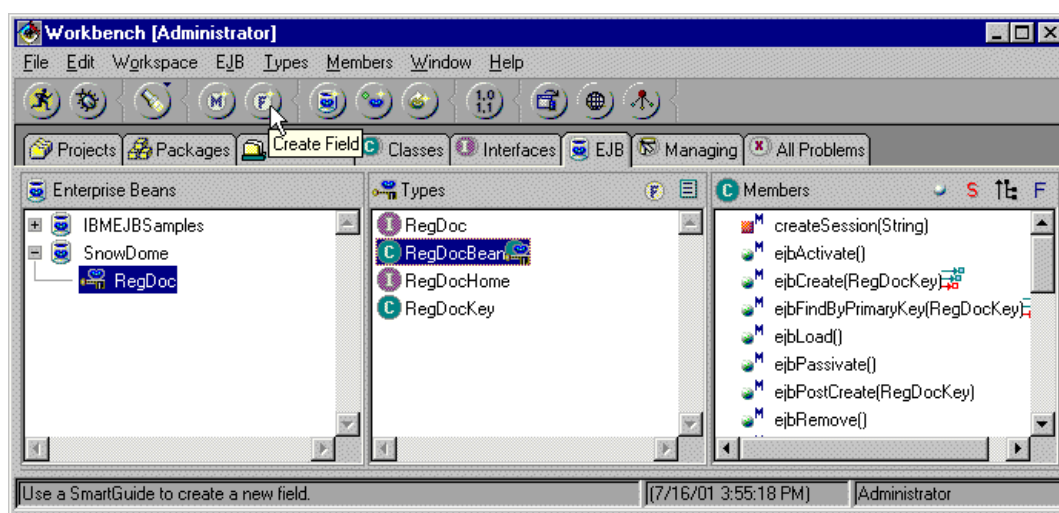


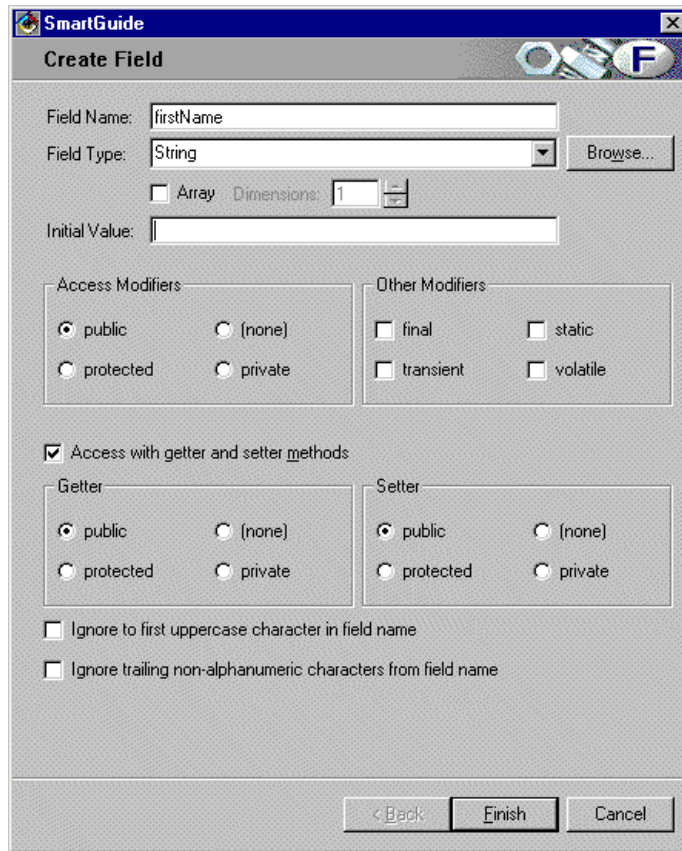
Figure 7-52 Create Field icon

2. Complete the SmartGuide window as shown in Table 7-7.

Table 7-7 Field values for Create Field SmartGuide

Field	Value
Field Name	firstName
Field Type	String
Access Modifiers	public
Access with getter and setter methods	Selected
Getter	public
Setter	public

3. The completed window should look like Figure 7-53. Click **Finish**.



The image shows the 'SmartGuide Create Field' dialog box. It has a title bar with 'SmartGuide' and a close button. The main area contains the following fields and options:

- Field Name:** A text box containing 'firstName'.
- Field Type:** A dropdown menu set to 'String', with a 'Browse...' button to its right.
- Array:** An unchecked checkbox, followed by 'Dimensions:' and a small numeric spinner set to '1'.
- Initial Value:** An empty text box.
- Access Modifiers:** A group box containing four radio buttons: 'public' (selected), 'protected', 'private', and '(none)'.
- Other Modifiers:** A group box containing four checkboxes: 'final', 'static', 'transient', and 'volatile', all of which are unchecked.
- Access with getter and setter methods:** A checked checkbox.
- Getter:** A group box containing four radio buttons: 'public' (selected), 'protected', 'private', and '(none)'.
- Setter:** A group box containing four radio buttons: 'public' (selected), 'protected', 'private', and '(none)'.
- Ignore to first uppercase character in field name:** An unchecked checkbox.
- Ignore trailing non-alphanumeric characters from field name:** An unchecked checkbox.

At the bottom, there are three buttons: '< Back', 'Finish', and 'Cancel'.

Figure 7-53 Create Field SmartGuide

The SmartGuide makes three modifications to the RegDocBean class.

- Declares a public variable called “firstName” of type String (java.lang.String)
- Creates a public getter method called “getFirstName”
- Creates a public setter method called “setFirstName”

4. Repeat the above process to add the fields listed in Table 7-8. Make sure to set the regNumber setter method to private.

Table 7-8 Fields to be added using the Create Field SmartGuide

Field name	Field Type	Access Modifiers	Getter Method	Setter Method
lastName	String	public	public	public
eMail	String	public	public	public
phone	String	public	public	public
company	String	public	public	public
why	String	public	public	public
regNumber	String	public	public	private

5. We now need to add fields to assist with the management of the Domino server connectivity. We add these directly to the RegDoc bean code.

From the Types pane of the EJB window, select **RegDocBean**. The bean definition code will be displayed in the Source pane. You will be able to see the declared variables for the fields created with the Create Field SmartGuide. We add four more variables as follows:

- private static Session session = null;
- private static Database database = null;
- private static View view = null;
- private Document doc = null;

6. Add these to the source as shown in Figure 7-54.

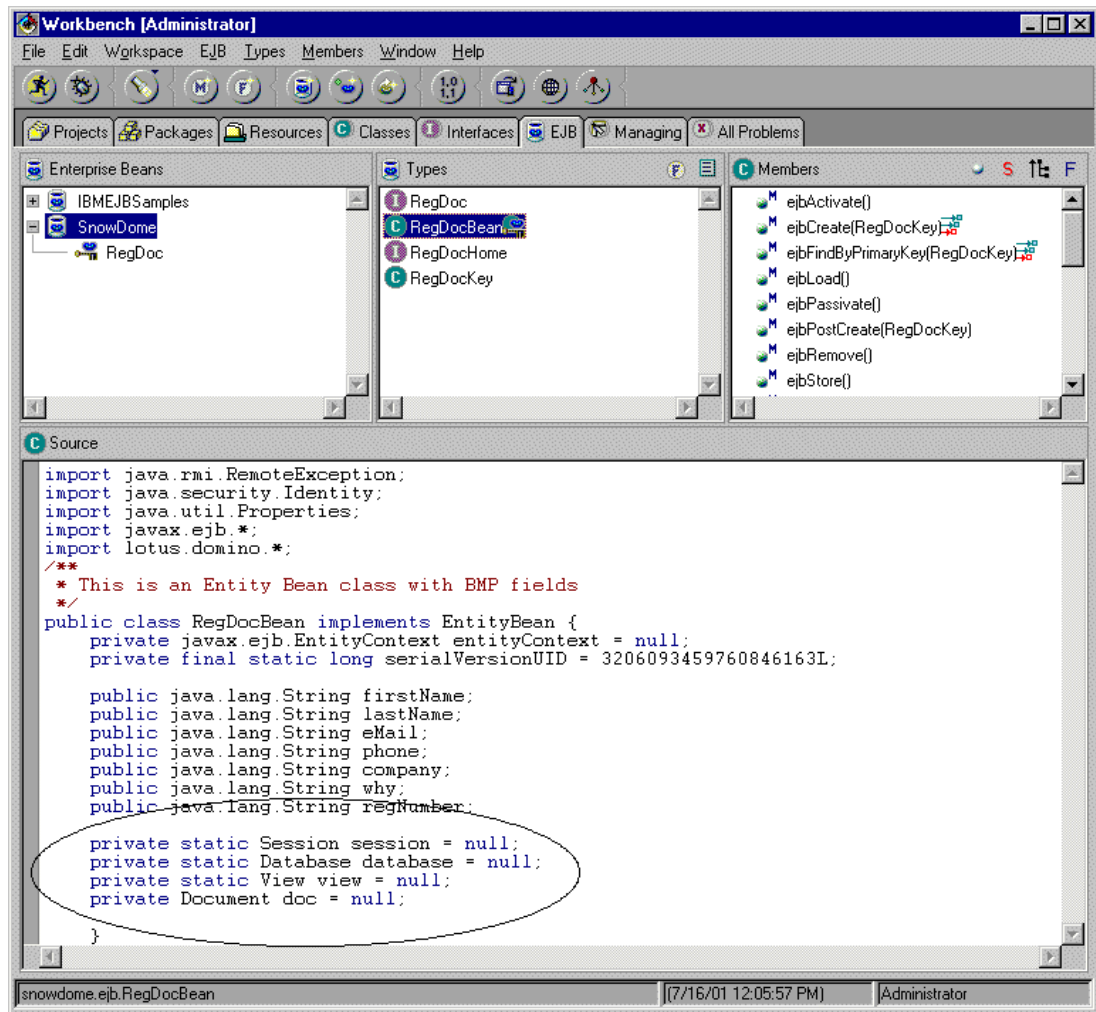


Figure 7-54 Source code for RegDocBean class definition

7. Save your changes.

We have now completed adding the required fields to the RegDocBean class.

## Adding custom methods

This section shows you how to add a custom method to the RegDoc EJB. We begin by creating a new method, createSession. This method will be responsible for managing the connectivity to the Domino server. To create the method, follow these steps:

1. All the methods we work with are part of the RegDocBean class. From the EJB window, select the **RegDocBean** class from the Types pane. The existing methods for the class will be displayed in the Members pane.
2. From the Types pane of the EJB window, select **RegDocBean**. Right-click to display the context menu and select **Add-> Method**. The Add Method SmartGuide appears.
3. On the first window select **Create a new method** and click **Next**. The attributes window is displayed. Complete the fields with the values shown in Table 7-9.

Table 7-9 Values for the createSession method

Field	Value
Method Name	createSession
Return Type	lotus.domino.Document
Access Modifiers	private
What parameters should this method have?	Add one parameter called "argCaller" of type java.lang.String
	Add another called "argEmail" of type java.lang.String
	Add another called "primaryKey" of type snowdome.ejb.RegDocKey

4. The completed SmartGuide is shown in Figure 7-55. Click **Finish** to create the method.

The screenshot shows the 'SmartGuide' window with the 'Attributes' tab selected. The 'Method Name' field contains 'createSession'. The 'Return Type' dropdown is set to 'lotus.domino.Document'. Under 'Access Modifiers', the 'private' radio button is selected. The 'Other Modifiers' section has 'final' checked. The 'What parameters should this method have?' list contains three entries: 'String argCaller', 'String argEmail', and 'RegDocKey primaryKey'. At the bottom, there are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

Figure 7-55 Add Method SmartGuide

5. We now need to add the code to the createSession method. To work with the method, select it from the Members pane. The source code will be displayed in the Source pane.



The `createSession` method has two primary functions. First it is used to manage the connectivity between WebSphere and Domino. Secondly, it returns the Domino document object to be used for the bean instance. The `create` and `finder` methods in the `RegDocBean` class will call this method before executing. Depending on the arguments passed into the method, a `Document` object is returned. If the `create` method is the caller, a new document is created. If the `finder` method calls, an existing document is located and referenced.

By declaring the session, database, and view objects as “static”, we are able to share the objects across all instances of the `RegDoc` bean. This means that our WebSphere EJB container has only one IIOP session open to Domino.

6. Figure 7-56 shows the code for the `createSession` method. When entering this code you will need to make some changes to suit your environment. The changes are as shown in Table 7-10.

*Table 7-10 Changes to the `createSession` method*

Example Code	Change to
<code>NotesFactory.getIOR("domwas11")</code>	<code>NotesFactory.getIOR("DOMINO_SERVER")</code>
<code>session.getDatabase("domwas11","registration.nsf")</code>	<code>session.getDatabase("DOMINO_SERVER","registration.nsf")</code>

**Note:** There is a lot of code to type over the next few sections. You may find it easier to cut and paste the source, or you can download the completed example. Refer to Appendix B, “Additional material” on page 295, for details.



```

private Document createSession(String argCaller, String argEmail, RegDocKey primaryKey)
{
    //We first test to see if a session obj has been created.
    if (session == null) {
        System.out.println("No Current Session. Lets create one. Caller is ->" +
argCaller);
        try {
            String ior = NotesFactory.getIOR("domwas11:8011");
            session = NotesFactory.createSessionWithIOR(ior);
            database = session.getDatabase("domwas11", "registration.nsf");
            view = database.getView("Attendees");
        }

        catch (NotesException e) {
            System.out.println("Error creating session.");
            e.printStackTrace();
        }
    }

    //Now we have a session we need to test if it is valid
    //If the session is invalid we re-set it to null.
    //The caller of the bean (the servlet) can then recall the bean method
    try {
        if (session != null) {
            if (argCaller == "ejbCreate" ) {
                doc = database.createDocument();
            }
            if (argCaller == "FindByEmail") {
                doc = view.getDocumentByKey(argEmail, true);
            }
            if (argCaller == "FindByPrimaryKey") {
                doc = database.getDocumentByID(primaryKey.primaryKey);
            }
        }
    }
    catch (Exception e) {
        e.printStackTrace();
        session = null;
        database = null;
        view = null;
        doc = null;
    }

    return doc;
}

```

Figure 7-56 Code for the createSession method -RegDocBean.java

This completes adding our custom method.

## Working with the callback methods

You will have noticed that VisualAge has generated several methods for the RegDocBean class. These are *callback methods* and are used as part of the EJB life cycle. As part of the EJB development process we need to put code into these methods.

In this topic we add code to the following callback methods:

- ▶ `ejbCreate`
- ▶ `ejbFindByPrimaryKey`
- ▶ `ejbLoad`
- ▶ `ejbStore`

### ***Modifying the `ejbCreate` method***

The `ejbCreate` method is used to create a new record in the persistent store. In our case this will be a new document in the registration database.

1. Modify the existing method by adding the code shown in Figure 7-57.

```
public RegDocKey ejbCreate(String argEmail) throws CreateException, RemoteException {

    //Inititalize all fields to null except email which is arg;
    eMail = argEmail;
    firstName = null;
    lastName = null;
    company = null;
    phone = null;
    why = null;
    regNumber = null;

    try {
        doc = createSession("ejbCreate", null, null);
        doc.save();
        regNumber = doc.getNoteID();
    }
    catch (NotesException e) {
        System.out.println("Exception while creating document");
        e.printStackTrace();
    }
    RegDocKey key = new RegDocKey(regNumber);
    return key;
}
```

Figure 7-57 `ejbCreate` method code

2. You will notice that we have changed the method signature. The parameter passed in has been changed from a `RegDocKey` to a `String`. When saving your changes, do a save replace rather than a save.

**Note:** If you save rather than save replace, you will end up with two create methods in the bean. Select the original method (`ejbCreate(RegDocKey)`) and delete it.

The create method must be added to the home interface of the bean. The methods on the home interface are used by EJB clients to create and find bean instances.

3. To add the `ejbCreate` method to the home interface, select it from the member's pane and right-click. Select **Add to -> EJB Home interface**.

When a method is added to the home interface, a symbol is displayed next to the method name (Figure 7-58).

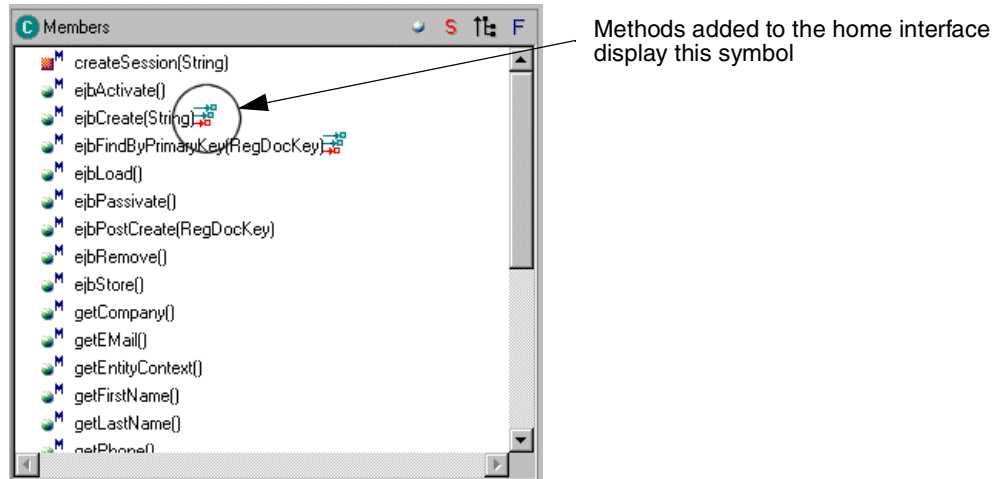


Figure 7-58 Members pane

### ***ejbFindByPrimaryKey***

The `ejbFindByPrimaryKey` method is used to retrieve a record from the persistent store. It uses the primary key of the bean to locate the record. In our case the method will return a `RegDoc` entity bean representing a Domino document. The primary key for our record will be the registration number, which is in effect the `NotelID` of the document.

1. Complete the `ejbFindByPrimaryKey` method as shown in Figure 7-59.

```
public snowdome.ejb.RegDocKey ejbFindByPrimaryKey(snowdome.ejb.RegDocKey primaryKey)
    throws java.rmi.RemoteException, javax.ejb.FinderException {

    doc = createSession("FindByPrimaryKey", null, primaryKey);

    // The Object Not Found Exception is caught by the servlet.
    if (doc == null)
        throw new ObjectNotFoundException("Unable to find Registration Document");
    return primaryKey;
}
```

Figure 7-59 `ejbFindByPrimaryKey` method code

2. You will notice that this method is included on the home interface by default.

### ***ejbLoad***

The `ejbLoad` method is responsible for keeping the bean data consistent with the persistent store. For our example, this method will locate the document in the registration database and set the bean fields accordingly.

Complete the code for the `ejbLoad` method as shown in Figure 7-60.

```

public void ejbLoad() throws java.rmi.RemoteException {
    try {
        //Included a call to createSession for debugging purposes
        RegDocKey key = (RegDocKey) getEntityContext().getPrimaryKey();
        doc = database.getDocumentByID(key.primaryKey);
        firstName = doc.getItemValueString("FirstName");
        lastName = doc.getItemValueString("LastName");
        company = doc.getItemValueString("CompanyName");
        phone = doc.getItemValueString("PhoneNumber");
        eMail = doc.getItemValueString("Email");
        why = doc.getItemValueString("WhySnowDomes");
        regNumber = key.primaryKey;
    }
    catch (NotesException e) {
        System.out.println("Error in ejbLoad method");
        e.printStackTrace();
    }
}

```

Figure 7-60 *ejbLoad method code*

### **ejbStore**

The `ejbStore` method updates the persistent store with the bean data. In our case, the user's registration document will be updated.

Modify the existing `ejbStore` method as shown in Figure 7-61.

```

public void ejbStore() throws java.rmi.RemoteException {
    // This method is called to update the data in domino with the beans data.
    try {
        //Call to create session included for debugging
        RegDocKey key = (RegDocKey) getEntityContext().getPrimaryKey();
        doc = database.getDocumentByID(key.primaryKey);
        doc.replaceItemValue("Form", "Registration");
        doc.replaceItemValue("FirstName", firstName);
        doc.replaceItemValue("LastName", lastName);
        doc.replaceItemValue("Email", eMail);
        doc.replaceItemValue("CompanyName", company);
        doc.replaceItemValue("PhoneNumber", phone);
        doc.replaceItemValue("WhySnowDomes", why);
        doc.save();
    }
    catch (NotesException e) {
        System.out.println("Error in ejbStore method");
        e.printStackTrace();
    }
}

```

Figure 7-61 *ejbStore method code*

We have now made all the required updates to the callback methods.

## Creating a custom finder method

An EJB finder method is a method used to locate records in the datastore. VisualAge for Java creates a default finder method called “`ejbFindByPrimaryKey`” for every entity bean it creates. This finder method locates an individual record in the store using the *key* to identify. EJBs can have multiple finder methods. This allows records to be located in the store via different parameters. In our example the key is the registration number, which is actually the document `NoteID`. Users may not remember their registration number, but they are likely to remember their e-mail address. To allow users to search by e-mail address, we create a new finder method called “`ejbFindByEmail`”.

1. To create the new finder method, select the current `ejbFindByPrimaryKey` method and edit as shown in Figure 7-62.
2. Since we have modified the method signature, saving it will create a new method.
3. We need to add the method to the home interface to allow EJB clients to use it. Refer to “Modifying the `ejbCreate` method” on page 230 for details on adding methods to the home interface.

```
public snowdome.ejb.RegDocKey ejbFindByEmail(java.lang.String argEmail)
    throws java.rmi.RemoteException, javax.ejb.FinderException {

    try {
        doc = createSession("FindByEmail", argEmail, null);
        // The Object Not Found Exception is caught by the servlet and handled
        if (doc == null)
            throw new ObjectNotFoundException("Unable to find Registration Document");
        regNumber = doc.getNoteID();
    }

    catch (NotesException e) {
        e.printStackTrace();
    }

    RegDocKey primaryKey = new RegDocKey(regNumber);
    return primaryKey;
}
```

Figure 7-62 *ejbFindByEmail* method code

## Promoting methods to the remote interface

This section shows how to promote enterprise bean methods to the remote interface.

The remote interface provides EJB clients with a way of calling EJB methods. Methods that are promoted to the remote interface are available for clients to call. The getter and setter methods of the EJB are usually available via the remote interface. For example, the `getLastName` method would be added to the remote interface so that EJB clients are able to retrieve the value of the `LastName` field of a bean.

In our example we promote the following fields to the remote interface:

- ▶ `getCompany`
- ▶ `getEmail`
- ▶ `getFirstName`
- ▶ `getLastName`
- ▶ `getPhone`
- ▶ `getRegNumber`
- ▶ `getWhy`

- ▶ setCompany
- ▶ setEmail
- ▶ setFirstName
- ▶ setLastName
- ▶ setPhone
- ▶ setWhy

Notice that we have not mentioned the setRegNumber method. This method should not be available to EJB clients. The value of the RegNumber field is managed by the Domino database. When a new document is created in the database, the NoteID is created automatically. The EJB does not need to set this value, nor should it be able to.

To promote the getCompany method:

1. Select it from the members pane. Right-click to display the context menu and select **Add to -> EJB Remote Interface**. Methods added to the remote interface display a symbol (Figure 7-63).

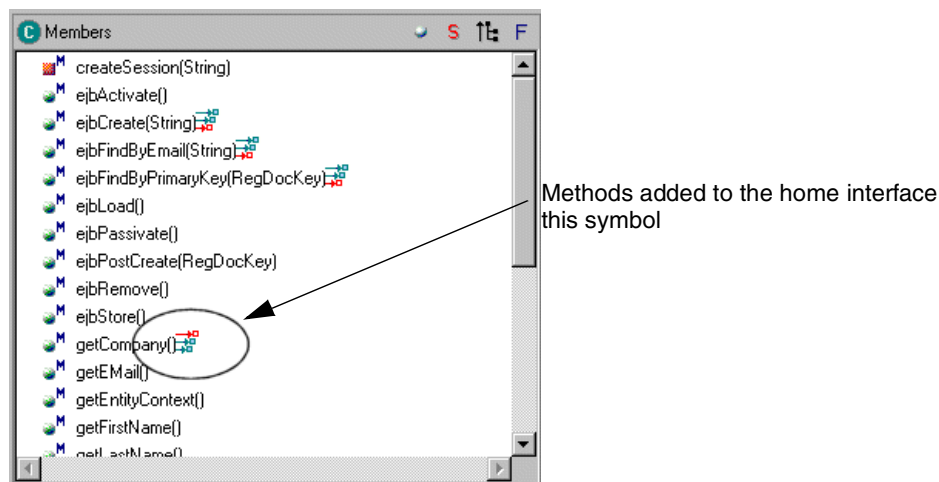


Figure 7-63 Methods added to the remote interface

2. Repeat the above process to add all above-mentioned methods to the remote interface.

## 7.4.2 Creating access beans and deployment code

This section shows how to create an access bean and the deployment code for the RegDoc bean.

### Access bean

An access bean is a JavaBean that contains the code needed to talk to an EJB. VisualAge includes an access bean wizard that allows you to generate all the code required. Once generated, access beans can be used like any other JavaBean. They can be placed on JSPs, used in servlets, or used in any other Java program. Access beans significantly reduce the complexity of EJB development. For detailed information about access beans, see *EJB Development with VisualAge for Java for WebSphere Application Server*, SG24-6144.

To create an access bean:

1. Select the **RegDoc** bean from the SnowDome EJB group of the Enterprise Beans pane.

2. Right-click to display the context menu and select **Add -> Access Bean**. This will display the Create Access Bean SmartGuide. Complete as shown in Table 7-11.

Table 7-11 Fields for Create Access Bean SmartGuide

Field	Value
EJB Group	SnowDome
Enterprise Bean	RegDoc
Access Bean Type	Copy Helper for an Entity Bean

3. Figure 7-64 shows the completed SmartGuide window. Click **Next**.

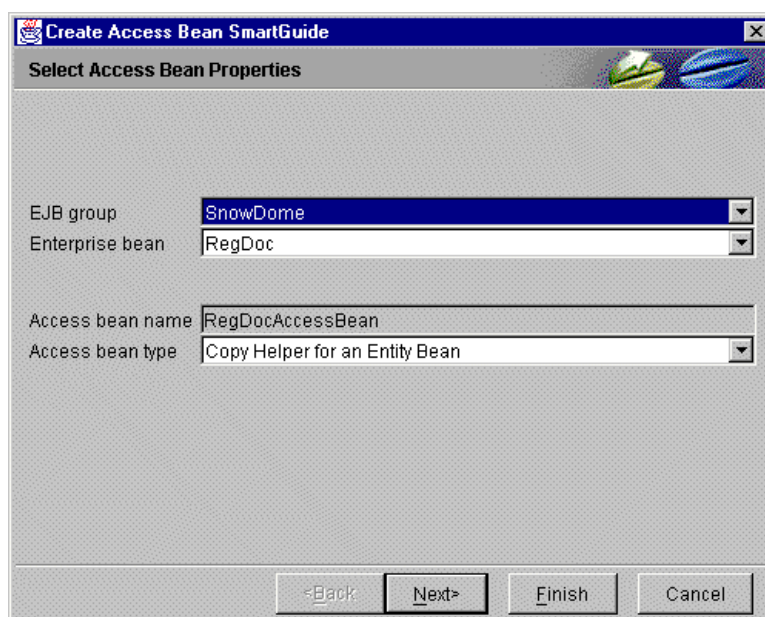


Figure 7-64 Select Access Bean Properties

4. On the Zero Argument Constructor window, leave the defaults and click **Next**.
5. The third window of the SmartGuide will be displayed. Leave the default values, and click **Finish**.
6. The SmartGuide will generate the access bean. When complete, the window as shown in Figure 7-65 will be displayed.

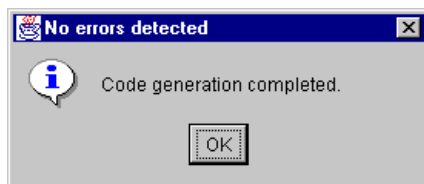


Figure 7-65 Successful creation of an access bean

## Deployment code

To generate the deployment code:

1. Select the **SnowDome** EJB group and right-click to display the context menu.
2. Select **Generate Deployment Code**. A window as shown in Figure 7-66 will be displayed.

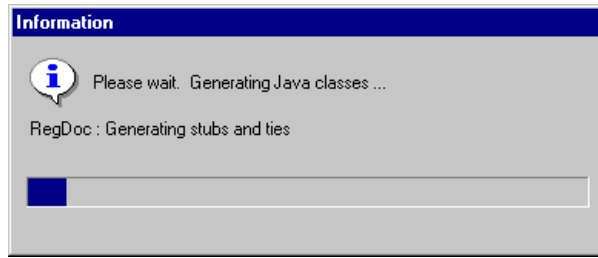


Figure 7-66 Generating the deployment code

Congratulations! You have successfully created a BMP bean for accessing Lotus Domino. The bean is now ready to be tested and deployed.

## 7.5 Application deployment

The first step in deploying our application is to do unit testing. This means testing that the EJB, servlets, JSPs, and HTML all function correctly as expected. To begin, we test the EJB using VisualAge for Java's EJB test client. When we have completed this, we then use VisualAge for Java to test all JSPs, servlets, HTML and EJBs. Our final step in deployment involves creating a new application server in WebSphere Application Server and using WebSphere Studio to publish the application.

### 7.5.1 Using the EJB test client

VisualAge provides a very powerful environment for testing enterprise beans. This topic will show you how to use the EJB test client to test the RegDoc bean.

The EJB test client is dependent on two services to work:

- ▶ A persistent name server.
- ▶ An EJB container to run the server side code.

VisualAge provides both of these services within its test environment. The persistent name server is part of the WebSphere Test Environment. The EJB container is part of the EJB development environment.

1. To start the Persistent Name Server, select **Workspace -> Tools -> WebSphere Test Environment**. This will launch the WebSphere Test Environment Control Center.
2. Select the Persistent Name Server from the left pane, and click **Start Name Server**. Figure 7-67 shows the Control Center with the persistent name server running.



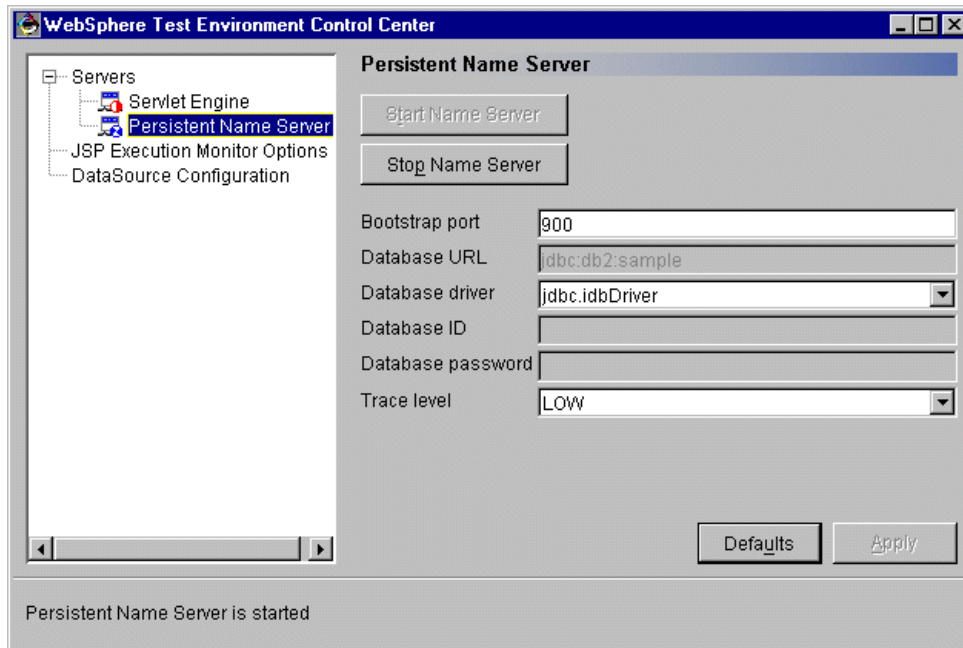


Figure 7-67 WebSphere Test Environment Control Center

3. Next start the EJB server container. From the EJB window, select the **SnowDome** EJB group and right-click. From the context menu select **Add To -> Server Configuration**. The EJB Server Configuration window will be displayed.
4. Click the **Start Server** icon from the toolbar. Figure 7-68 shows the EJB Server Configuration window with the Start Server icon highlighted.

**Note:** When starting the persistent name server and the EJB server, you must start the persistent name server first. Do not start the EJB server until the persistent name server has completely started. You can check that it has finished loading by looking at the VisualAge console. When the console displays the message `E Server open for business`, the persistent name server has started.

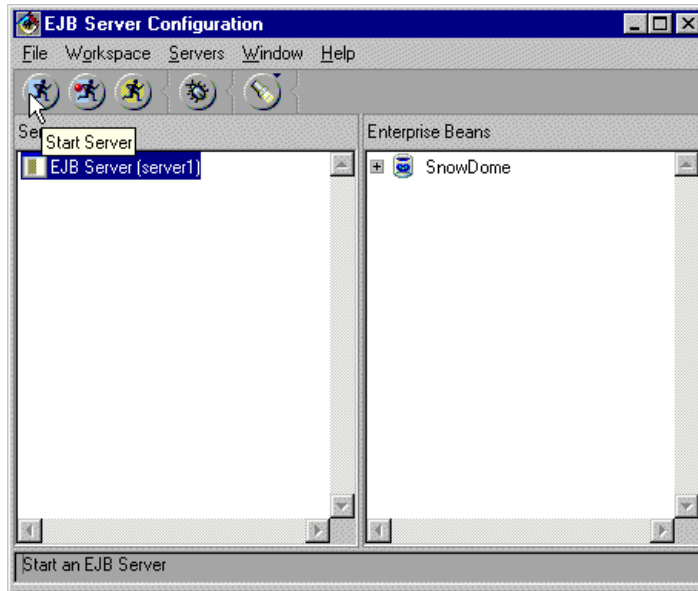


Figure 7-68 EJB Server Configuration window

5. Now that the persistent name server and EJB server are both running, we are able to launch the test client. Select the **SnowDome** EJB group and right-click. From the context menu, select **Run Test Client**. The EJB Test Client window is displayed as shown in Figure 7-69.
6. Click **Lookup**.

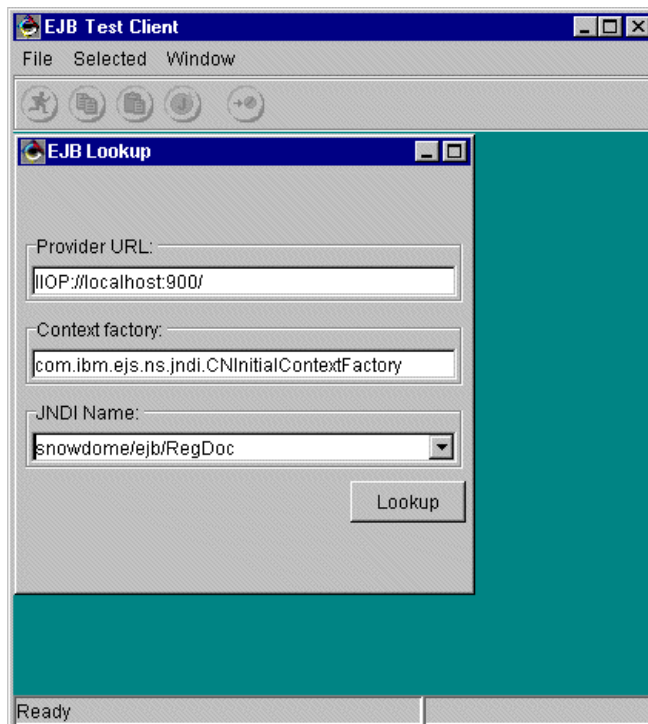


Figure 7-69 EJB Test Client Lookup window

7. The EJB test client will display the RegDoc bean as shown in Figure 7-70.

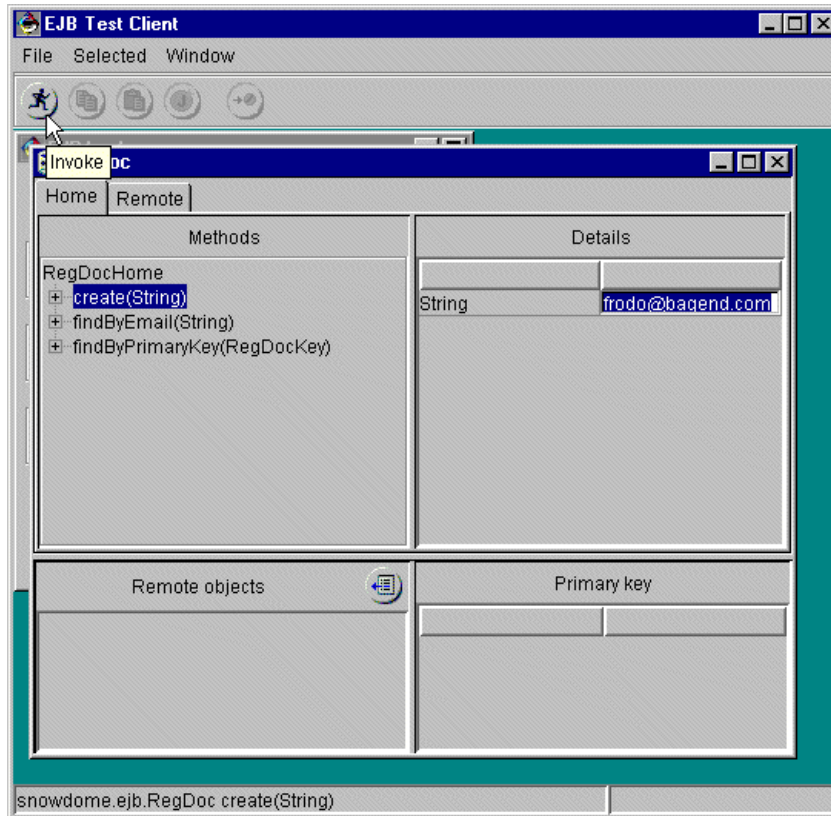


Figure 7-70 EJB Test Client home interface window

The window displays the Home tab. This represents the home interface of the bean. The methods that we added to the home interface are visible here.

8. Test our bean by selecting the **create** method.
9. For the String parameter, enter johnsmith@abccompany.com. Once entered, select the **Invoke** icon from the toolbar.
10. The test client creates a bean instance and displays the remote interface. Figure 7-71 shows the EJB test client with the remote interface methods.

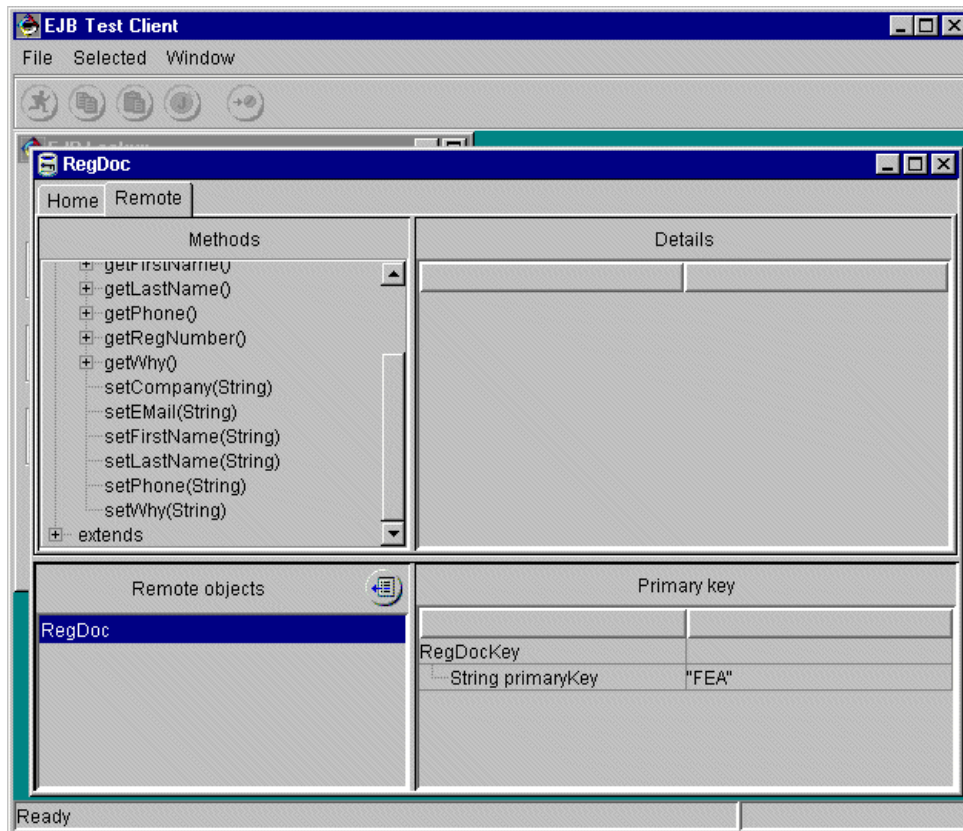


Figure 7-71 EJB Test Client Remote interface methods

**Note:** If your EJB does not test correctly, there are a number of things you should check. First make sure you meet the following requirements:

- ▶ Your Domino server is accessible over the network via the name specified in the createSession method.
- ▶ Your Domino server is running both HTTP and IIOP.
- ▶ You have the registration database on the Domino server, and it is fully indexed.
- ▶ You have started the persistent name server in the WebSphere Test Environment.
- ▶ You have started the EJB server.
- ▶ You have copied the code exactly as displayed in the examples (with the exception of the server name).
- ▶ You have promoted the correct methods to the home and remote interfaces.

If you are sure you have everything correct, you should use the VisualAge debugger to determine where the error is coming from. You are able to place breakpoints in any of the methods of your bean and step through the code.

When testing, you may need to make code changes. You can do this without having to stop and start the EJB server. However, if you add or remove methods from the home or remote interfaces, you must stop the EJB server and re-generate the deployment code and access bean.

11. We now invoke each of the setter methods to update the Domino document. From the Methods pane, select **setCompany**. In the Details pane, enter Morder Destruction.
12. Select the **Invoke** icon from the toolbar.
13. Repeat this process for the methods listed in Table 7-12.

Table 7-12 Sample data for testing the remote methods

Method	Value
setFirstName	John
setLastName	Smith
setPhone	123-4567
setCompany	<Already Called>
setEmail	johnsmith@abccompany.com

14. Complete the testing by opening the Domino database, registration.nsf, and looking at the newly created document.
15. Using the EJB test client, switch back to the Home interface tab. Invoke the findByEmail method. Use the e-mail address johnsmith@abccompany.com. Test the getter methods of the bean. You should be able to retrieve all the information entered in the previous steps.

## 7.5.2 Using the WebSphere Test Environment

This section covers using the WebSphere Test Environment to run the entire Snow Dome application. Referring back to the application architecture in Figure 7-1, we are now ready to test the entire application. Four extra files are needed to complete testing:

- ▶ sdSearch.jsp
- ▶ controller.class
- ▶ sdsucessupdate.jsp
- ▶ regejb.html

These files are listed in Appendix A, “Code examples” on page 265. You can create these files as listed, or download the entire application as described in Appendix B, “Additional material” on page 295.

1. To run the application, place the JSPs and HTML files in the following directory:  
`X:\VAJAVA_INSTALL_DIRECTORY\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web`
2. If you have downloaded the source for the controller servlet, import it into VisualAge. For details on importing the code, refer to Appendix B, “Additional material” on page 295.
3. Launch the WebSphere Test Environment and start the persistent name server. For details refer to Section 7.5.1, “Using the EJB test client” on page 236.
4. Launch the EJB server manager and start the EJB server. For details, refer to Section 7.5.1, “Using the EJB test client” on page 236.
5. Start the WebSphere Test Environment Servlet Engine. Refer to “Testing” on page 206 for details.
6. Launch a Web browser and enter the following URL:

`http://localhost:8080/SDIndex.html`

The Snow Dome World 2002 entry window will be displayed as shown in Figure 7-72.

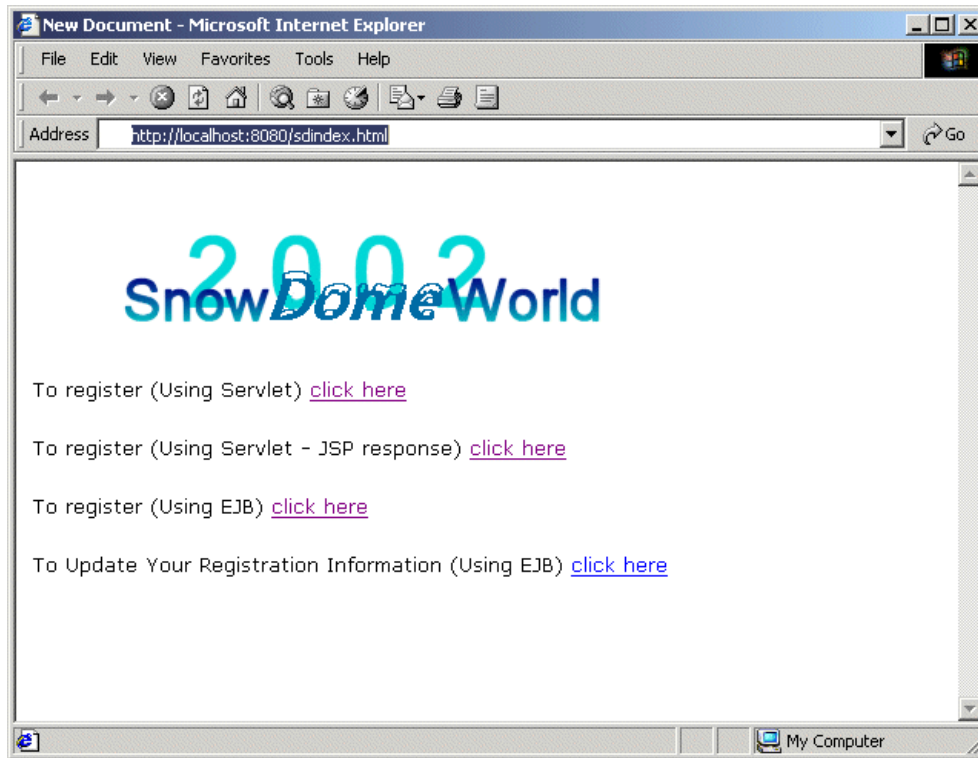


Figure 7-72 Snow Dome World entry window

7. Test each section of the site. If errors occur, or the site does not behave as expected, use the VisualAge debugging tools to address the error. Use the VisualAge for Java console to help locate problems.

### 7.5.3 Deployment to WebSphere Application Server

This section details how to deploy the Snow Dome application to WebSphere Application Server and the Lotus Domino Server. First, you must meet the following criteria:

- ▶ Have access to the OS/400 Integrated File System (IFS) to publish files.
- ▶ Have the Domino and WebSphere servers installed and accessible.
- ▶ Have WebSphere Studio V3.5 installed.
- ▶ Have the Domino remote classes configured in WebSphere. Refer to Section 6.3, "Configuring a WebSphere Application Server" on page 156.
- ▶ Have completed all of the examples in this chapter or downloaded the example files as detailed in Appendix B, "Additional material" on page 295.

To deploy the application involves the following steps:

1. Generate JAR files in VisualAge.
2. Use WebSphere Studio to publish the files.
3. Create an application server in WebSphere Application Server
4. Register the EJB and modify the classpath.
5. Testing.

## Generating the JAR files

A JAR file is an archive file that consists a group of related Java class files. The advantage of using a JAR file is that the number of files required to deploy an application is significantly reduced. In our application we create three JAR files. One is used to deploy the servlets, another to deploy the EJB client code, and the final one is used to deploy the EJB bean. To create the JAR files follow these steps:

1. From VisualAge for Java, go to the EJB window of the Workbench. Right-click the **SnowDome** project, and select **Export -> Generate Deployed Jar**. Refer to Table 7-13 for values.

Table 7-13 Values for the Export to a Deployed JAR file SmartGuide

Field	Value
JAR file	c:\temp\sdejb.jar
Beans	Selected
Class	Selected
Java	None
Resource	Selected

2. Figure 7-73 shows the completed SmartGuide. Click **Finish**.

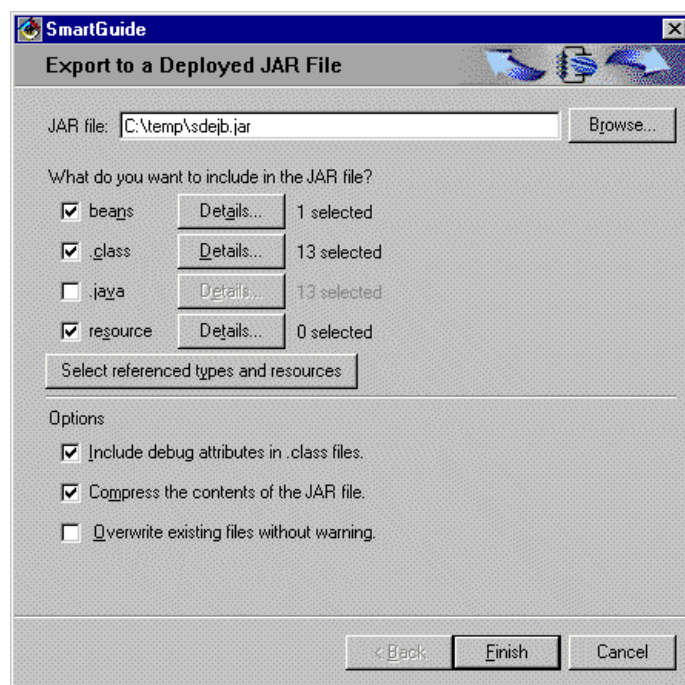


Figure 7-73 Export to Deployed JAR file

3. Now we will create the JAR file for the EJB client code. Right-click the **SnowDome** group, and select **Export -> Client JAR**. Complete the fields as shown in Table 7-14.

Table 7-14 Values for Export to EJBClient JAR file SmartGuide

Field	Value
JAR file	c:\temp\sdclient.jar

Field	Value
Beans	Selected
Class	Selected
Java	None
Resource	Selected

Figure 7-74 shows the completed SmartGuide.

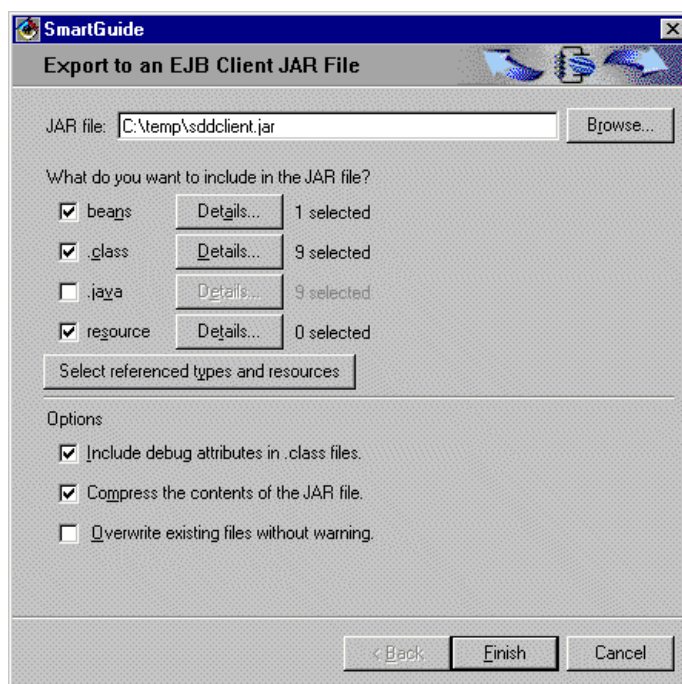


Figure 7-74 Export to EJB Client jar file SmartGuide

- We now need to create one more JAR file. This file will contain the servlets used by the application. Change the Workbench window to the projects window. Expand **Snow Dome Project**, and select the package **somedome.servlets**. Right-click and select **Export**. The Export to a JAR file SmartGuide starts.
- From the first window select **JAR file**, and click **Next**. Complete the second window as shown in Table 7-15.

Table 7-15 Field values for the Export to a JAR File SmartGuide

Field	Value
JAR file	c:\temp\sdservlets.jar
Class	Selected
Java	Not selected
Resource	Selected
beans	Not selected

- Figure 7-75 shows the completed SmartGuide. Click **Finish** to create the JAR file.



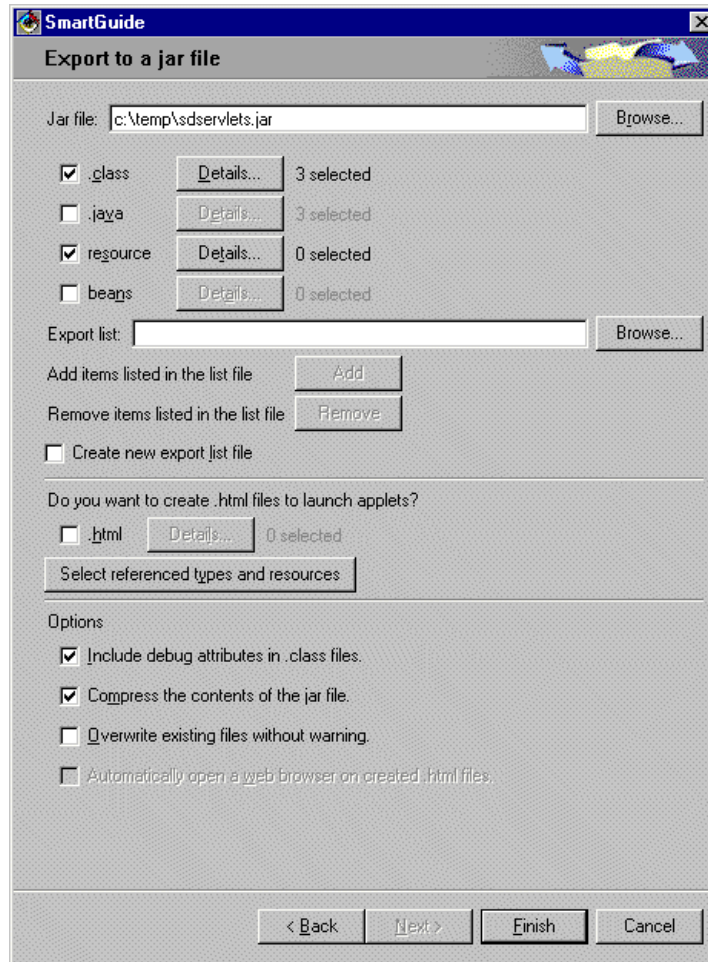


Figure 7-75 Creating the servlet JAR file

We have created the three JAR files needed to deploy the application's Java code.

## Publish using WebSphere Studio

WebSphere Studio acts as a central management point for all Web application components. It has tight integration with both VisualAge for Java and WebSphere Application Server. Using this tool we are able to migrate our code from VisualAge to WebSphere. A complete discussion on WebSphere Studio is beyond the scope of this book. An excellent reference is *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, SG24-5755.

1. Start WebSphere Studio and create a new project called "SnowDome".
2. Right-click the SnowDome project and select **Insert -> File** as shown in Figure 7-76. The Insert File window is displayed.

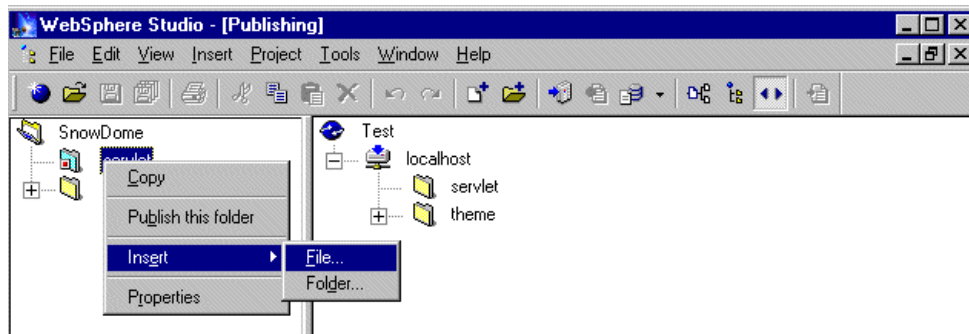


Figure 7-76 Inserting new files into a Studio project

3. From the Insert File window, select the **Use Existing** tab.
4. Browse to the VisualAge directory where all the GIF, HTML and JSP files are located. This directory is as follows:  
X:\VAJAVA\_INSTALL\_DIRECTORY\ide\project\_resources\IBM WebSphere Test  
environment\hosts\default\_host\default\_app\web
5. Using the CTRL key select all files, and click **OK**. Figure 7-77 shows the Insert File window after selecting all files. Click **OK**.

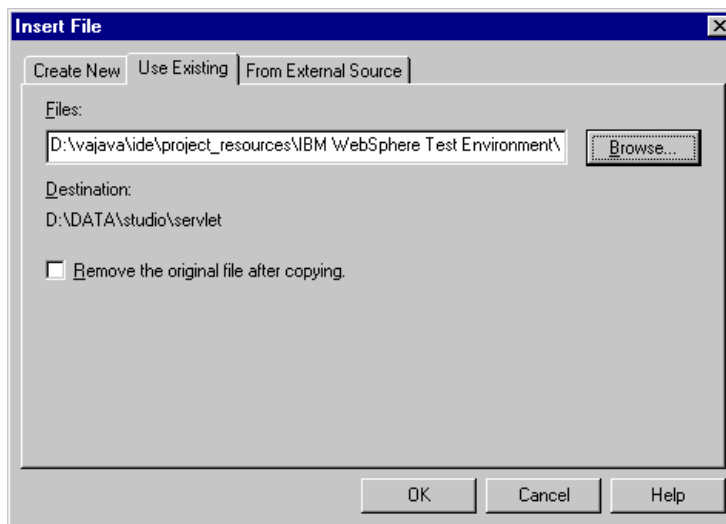


Figure 7-77 Adding all files

6. Figure 7-78 shows the WebSphere Studio environment with the HTML, GIF and JSP files included.

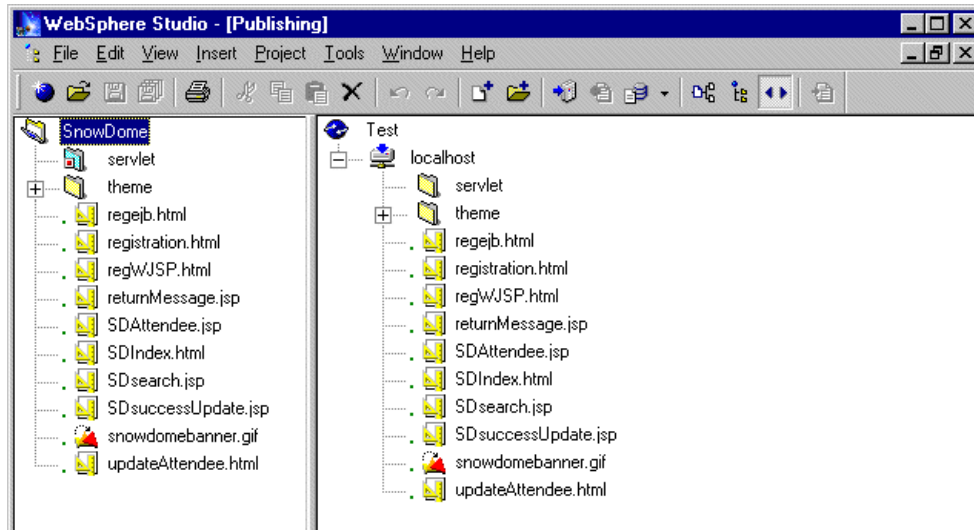


Figure 7-78 WebSphere Studio with the HTML, GIF and JSP files

7. From WebSphere Studio, select the **Snow Dome** project. Select **Insert -> Folder**. Enter the folder name as “classes” and click **OK**. We will use the classes folder to store the sdclient.jar.
8. Repeat this step to create another folder called “ejb”. This folder will store the sdejb.jar. WebSphere Studio includes a folder called “servlet” by default. We will use this folder to store our sdservlets.jar file.
9. We now need to add the JAR files created in “Generating the JAR files” on page 243. From WebSphere Studio select the newly created classes folder.
10. From the context menu select **Insert -> File**. The Insert File window is displayed.
11. Select the **Use Existing** tab, and browse to the C:\temp directory.
12. Select the sdclient.jar file.
13. Repeat this step and add the sdejb.jar file to the ejb folder.
14. Repeat again to add the sdservlets.jar to the servlet folder.

Figure 7-79 shows the WebSphere Studio environment after creating the folders, and importing the JAR files.

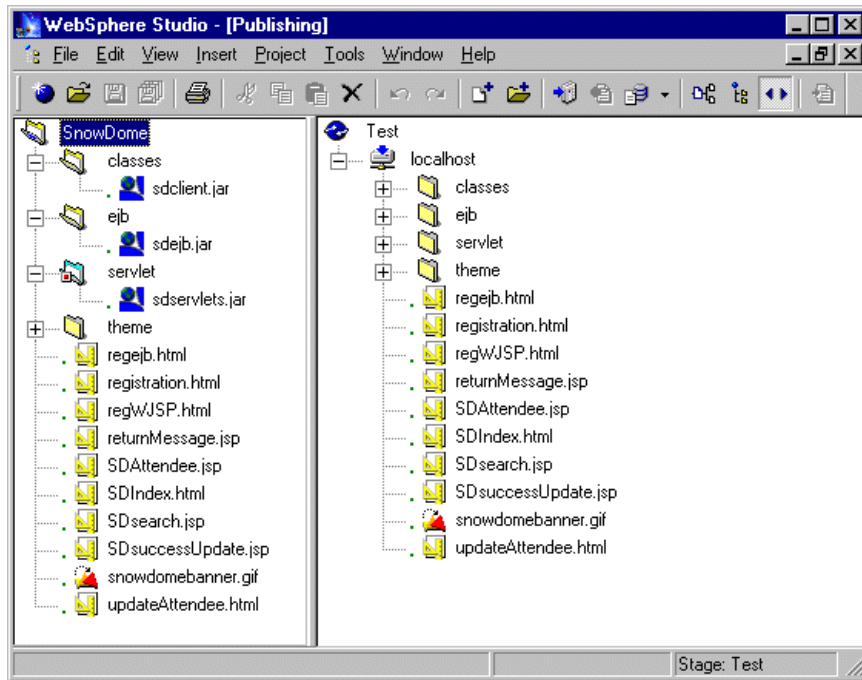


Figure 7-79 Imported JAR files

15. Now that we have the application components in WebSphere Studio, we need to tell WebSphere Studio where to put the files when we publish. To do this, we create some publishing targets, and then assign a directory to each target.

From WebSphere Studio, select **localhost** from the right pane. Right-click to display the context menu and select **Properties** (Figure 7-80).

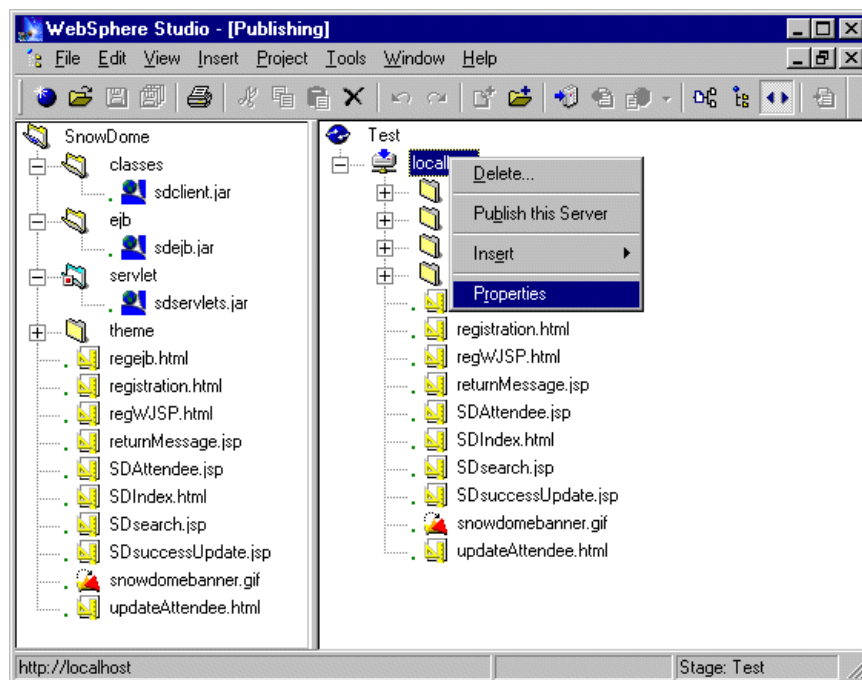


Figure 7-80 Context menu for the localhost server

16. The localhost Properties window will be displayed (Figure 7-81). Click **Targets**. The Publishing Targets window is displayed.

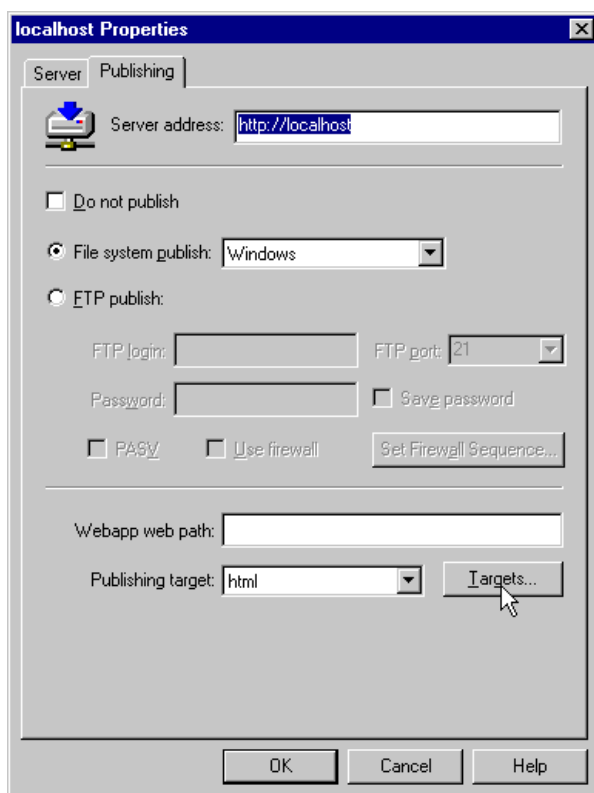


Figure 7-81 localhost Properties window

17. We need to add two new targets, and modify the targets for both servlet and HTML. Table 7-16 shows the directories for each target. You should change the drive letter to whatever letter you have the QIBM directory of your iSeries server mapped. Also, make sure you change the WebSphere Application Server instance name to match your environment. In our example the WebSphere Application Server instance name was WAS11.

Table 7-16 Publishing targets

Target	Destination
classes	I:\UserData\WebASAdv\WAS11\hosts\default_host\sdWebApp\classes
ejb	I:\UserData\WebASAdv\WAS11\deployedEJB
servlet	I:\UserData\WebASAdv\WAS11\hosts\default_host\sdWebApp\servlets
HTML	I:\UserData\WebASAdv\WAS11\hosts\default_host\sdWebApp\web

Figure 7-82 shows the publishing targets window as we completed it.

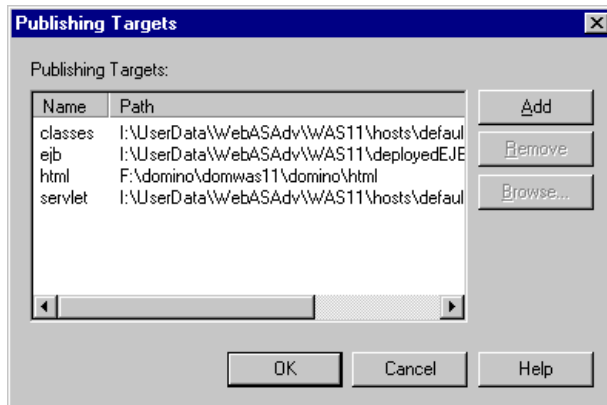


Figure 7-82 Publishing Targets window

18. Now we have created the publishing targets we need to assign a target to each of the folders we created.
19. Select the folder (for example **classes**) from the right-hand pane of WebSphere Studio. Right-click and select **Properties**. The classes Properties window is displayed.
20. Complete as shown in Figure 7-83. Make sure you select **Make this folder a virtual directory**. Click **OK**.

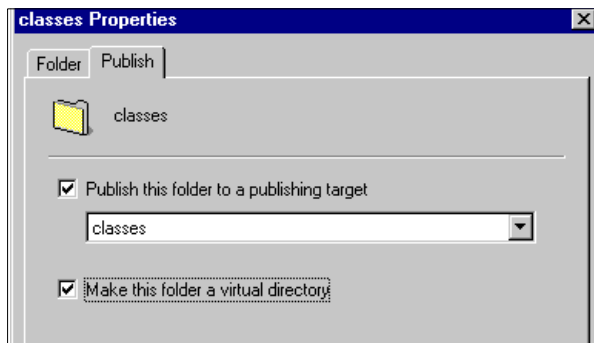


Figure 7-83 Classes properties

21. Repeat the above steps and assign the targets for each folder. Figure 7-17 shows the mappings for the other folders.

Table 7-17 Folder to target mapping

Folder	Target
servlet	servlet
ejb	ejb

22. Before we can publish our files we need to create the directories on the iSeries server.

On your iSeries server create the following directories. Make sure you substitute WAS11 with your WebSphere Application Server instance name.

- /QIBM/UserData/WebASAdv/WAS11/hosts/default\_host/sdWebApp
- /QIBM/UserData/WebASAdv/WAS11/hosts/default\_host/sdWebApp/servlets
- /QIBM/UserData/WebASAdv/WAS11/hosts/default\_host/sdWebApp/classes
- /QIBM/UserData/WebASAdv/WAS11/hosts/default\_host/sdWebApp/web

23. We are now ready to publish our files.

24. In WebSphere Studio click **SnowDome** and select **Publish Whole Project**. The Publishing Options window is displayed.
25. Leave the defaults and select **OK**.
26. You may get a window pop-up informing you that some files have broken child links. Ignore this and click **OK**. Figure 7-84 shows the progress indicator you will see as the project is published.

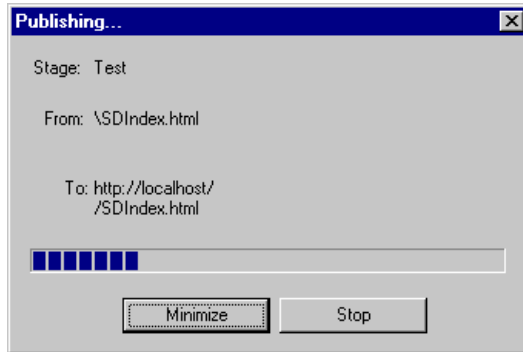


Figure 7-84 Publishing the files

We have now completed publishing the files to the iSeries server. We now need to create an application server to house the Snow Dome application.

### Creating the application server

Perform the following steps to create an application server:

1. Start the WebSphere Administration Console.
2. Select the Wizard icon and **Create Application Server** (Figure 7-85) This starts the Create Application Server wizard.



Figure 7-85 Starting the Create Application Server wizard

3. Complete the first window as displayed in Figure 7-86. Click **Next**.

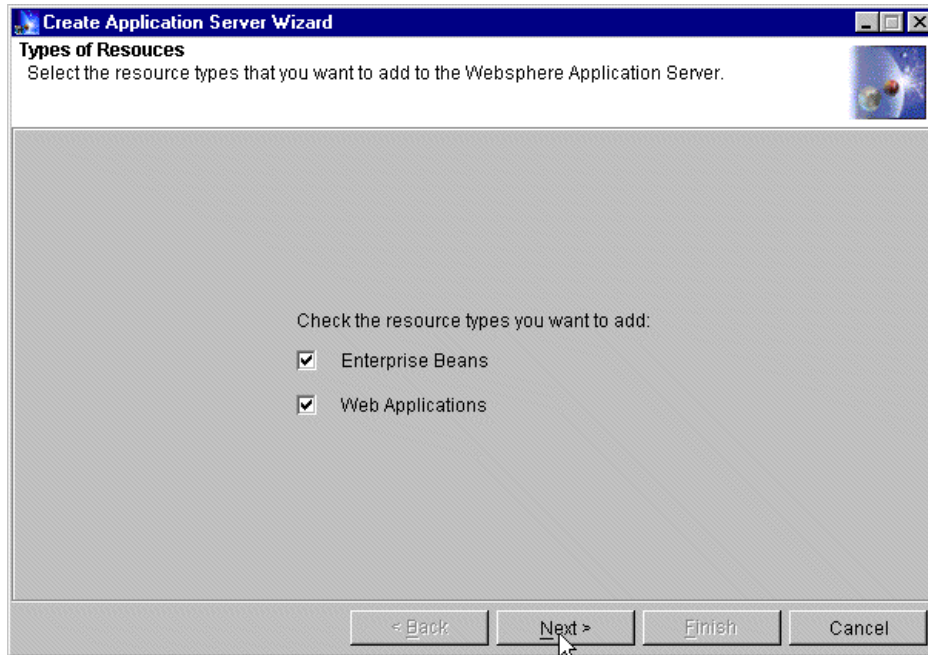


Figure 7-86 Create Application Server Wizard window 1

4. Complete the Application Server Properties window as shown in Table 7-18.

Table 7-18 Arguments for the Create Application Server properties

Field name	Value
Application Server Name	SnowDome Server
Command line arguments	-Xms32m -classpath /QIBM/ProdData/lotus/notes/data/domino/java/NCSOW.jar
Standard output	/QIBM/UserData/WebASAdv/WAS11/sdstdout.txt
Standard error	/QIBM/UserData/WebASAdv/WAS11/sdstderr.txt

5. Figure 7-87 shows the completed window. Click **Next**.



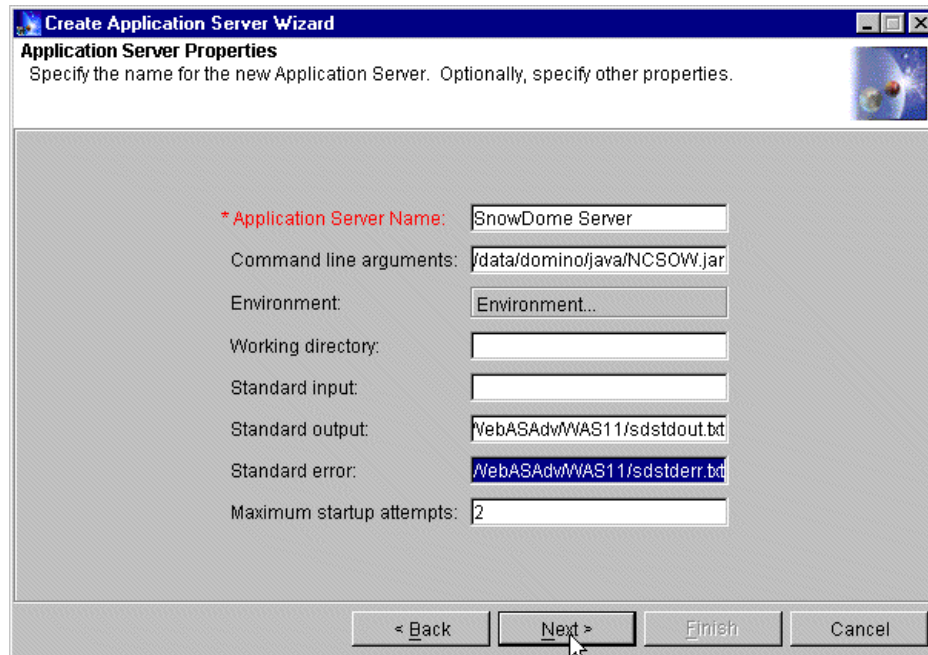


Figure 7-87 Create Application Server Properties

6. On the Application Server Start Option window, change the value to **Start the server automatically after creating it** (Figure 7-88). Click **Next**.

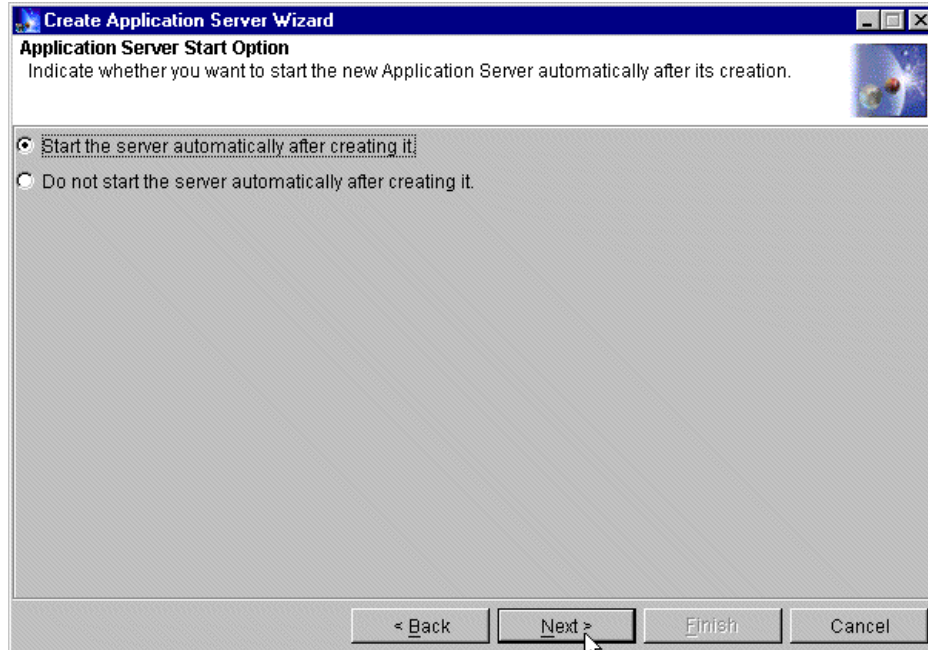


Figure 7-88 Application Server Start Option

7. Select your node on the Node Selection window (Figure 7-89) and click **Next**.

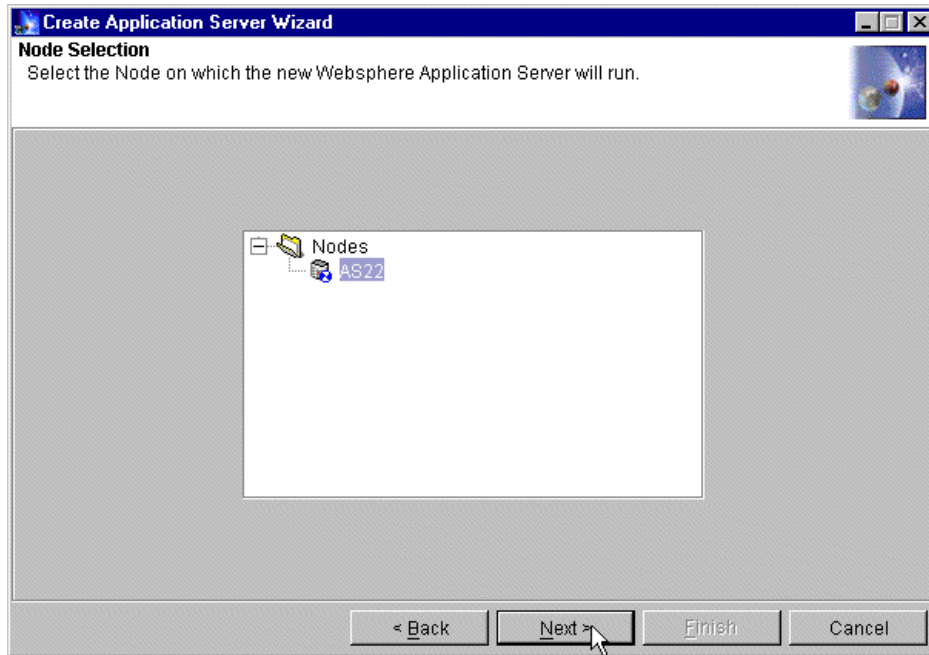


Figure 7-89 Node selection

8. On the Add Enterprise Beans window, select **Browse** (Figure 7-90).

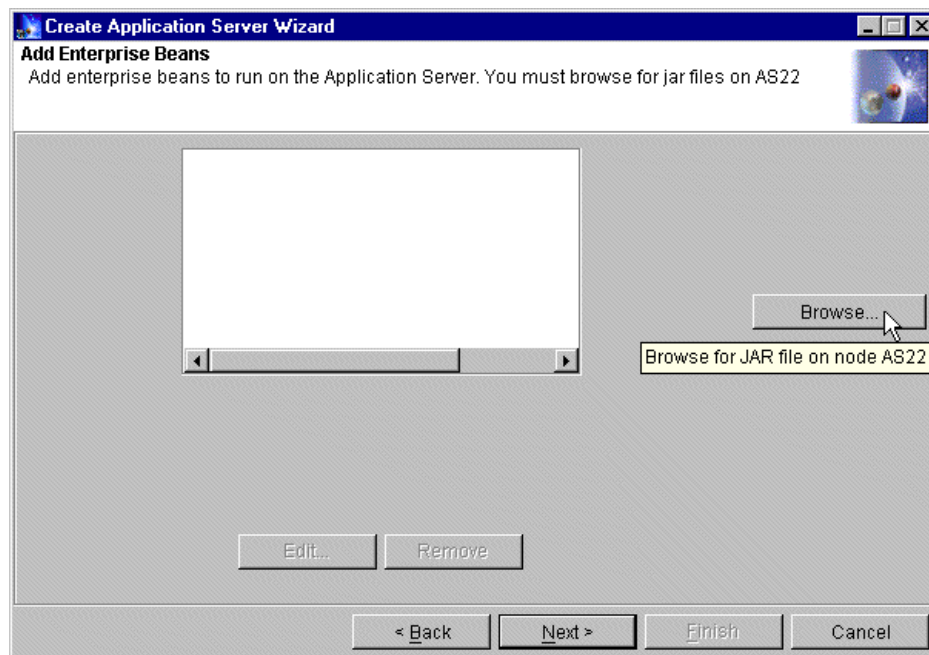


Figure 7-90 Add Enterprise Bean

9. Navigate to the deployedEJB directory and select **sdejb.jar** (Figure 7-91) Click **Select**.

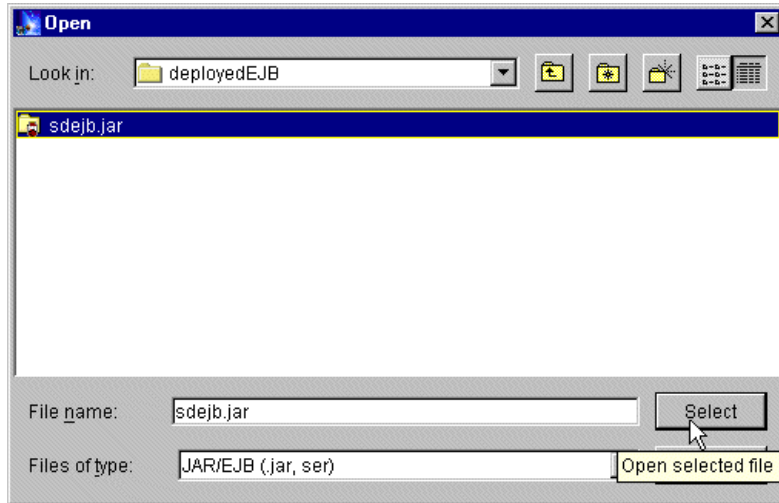


Figure 7-91 Selecting the sdejb.jar file

10. Click **Yes** on the Confirm window that is displayed (Figure 7-92).

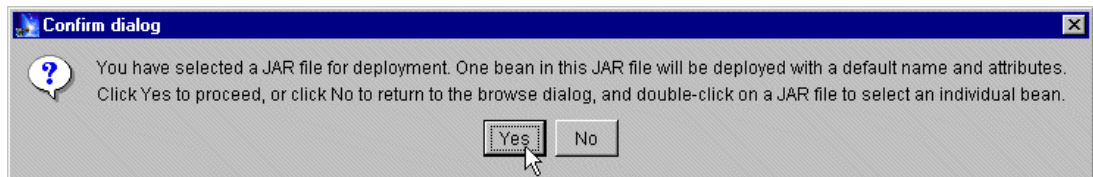


Figure 7-92 Confirm window

11. You will return to the Add Enterprise Beans window, which will now include an entry for the RegDoc bean. Click **Next**.
12. The EJB Container properties pane is displayed. Leave the defaults and click **Next**.
13. The Select Virtual Host window is displayed (Figure 7-93). Select **default\_host** and click **Next**.

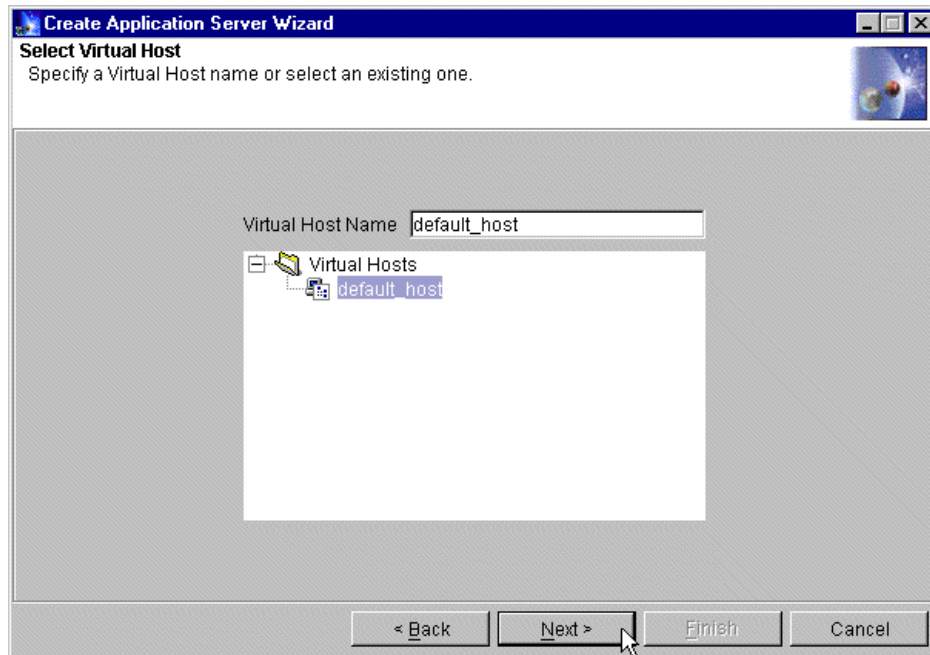


Figure 7-93 Select Virtual Host

14. The Servlet Engine Properties window is displayed. Leave this as default and click **Next**.
15. The Web Application Properties window is displayed. Change the Web Application Name to sdWebApp as shown in Figure 7-94 and click **Next**.

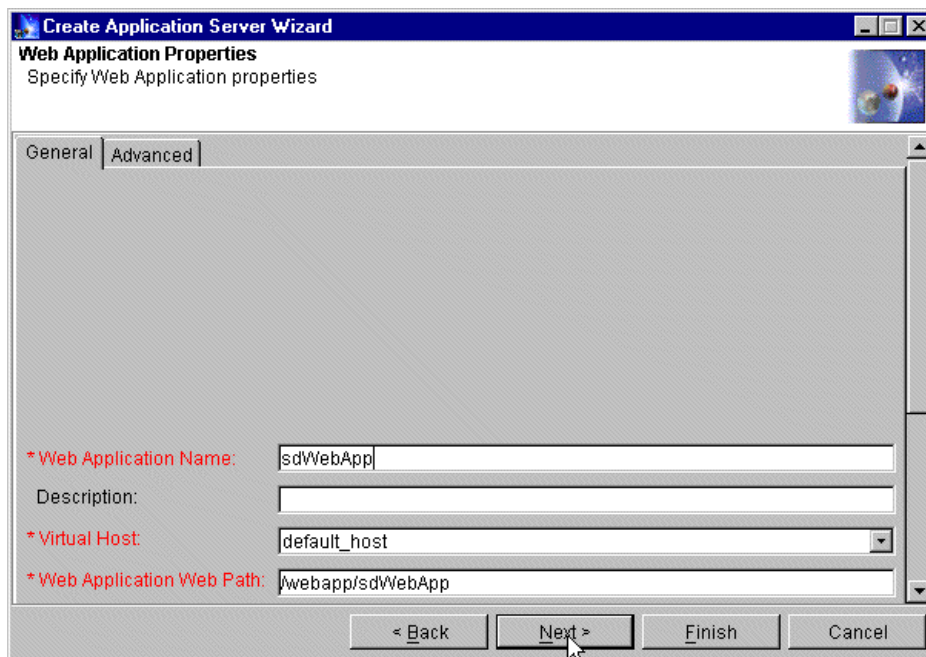


Figure 7-94 Web Application Properties

16. The Specify System Servlets window is displayed. Complete as shown in Figure 7-94. Click **Finish**.

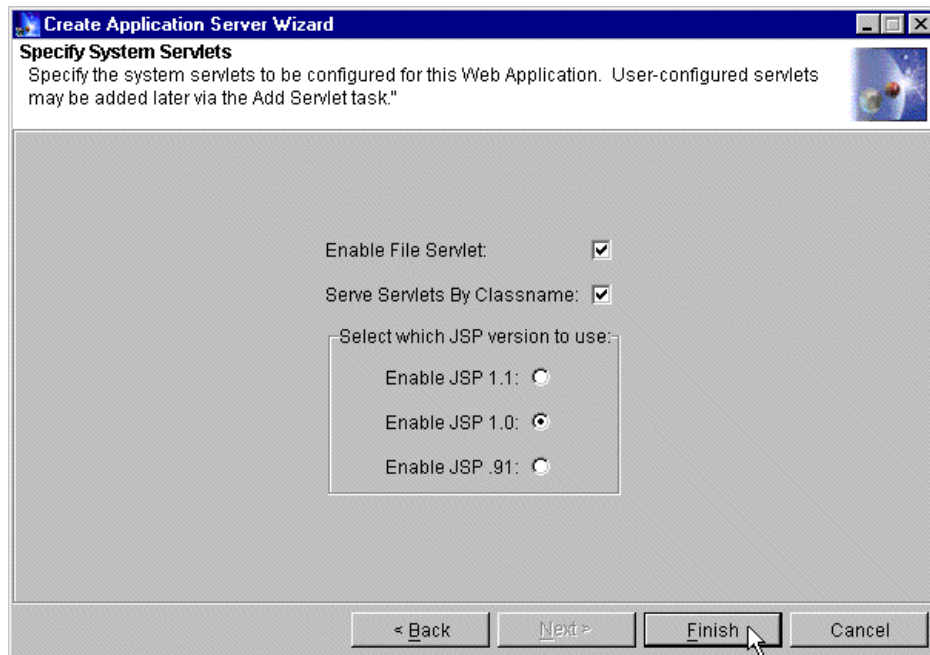


Figure 7-95 Specify System Servlets

This completes creating the application server we use to run the Snow Dome Application Server.

## Modifying Web Application classpath

In this section we modify the classpath of the SnowDome server. This is necessary to allow the application server to locate the class files needed for the Snow Dome application.

1. Start the WebSphere Administration Console.
2. Expand the node, then **SnowDome Server**, and then **SnowDome Server ServletEngine**.
3. Select **sdWebApp** and click the **Advanced** tab. Figure 7-96 shows the sdWebApp Advanced window.

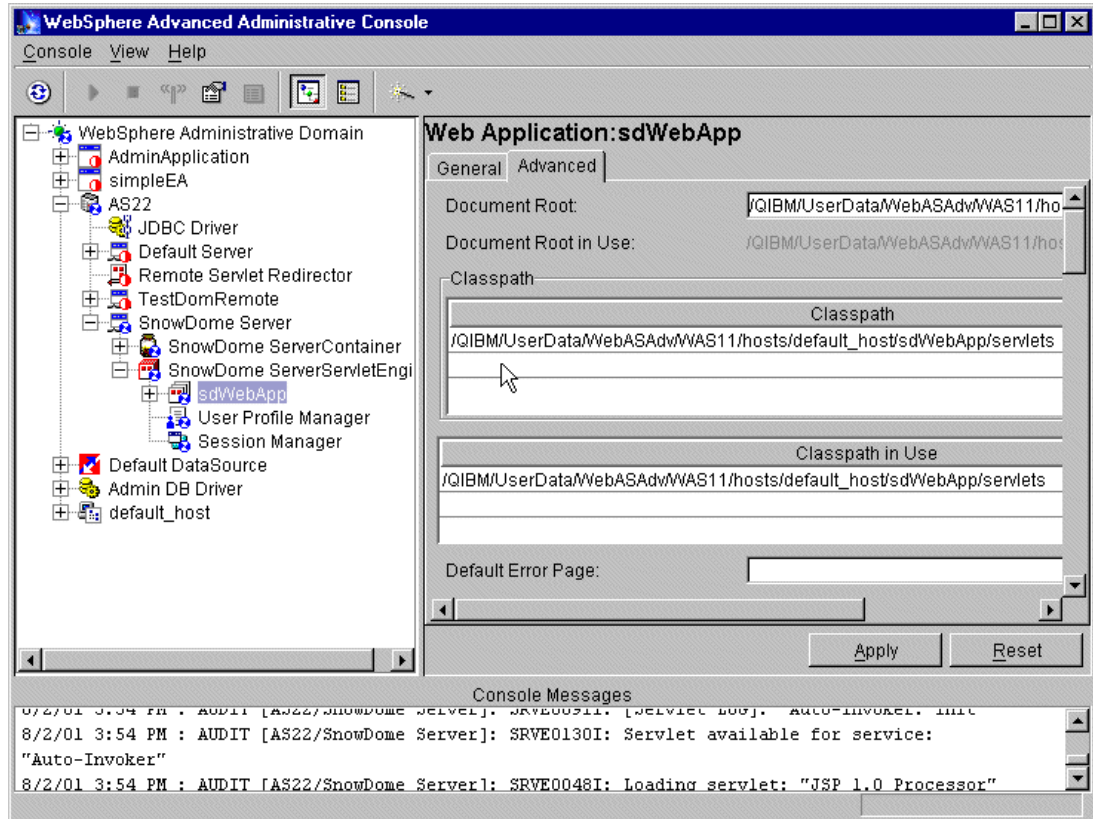


Figure 7-96 Selecting the sdWebApp - Advanced tab

4. Add the following classpaths. Make sure to replace the WebSphere Application Server instance name WAS11 with your instance name.

```
/QIBM/UserData/WebASAdv/WAS11/hosts/default_host/sdWebApp/servlets/sdservlets.jar
/QIBM/UserData/WebASAdv/WAS11/hosts/default_host/sdWebApp/classes/sdclient.jar
/QIBM/ProdData/lotus/notes/data/domino/java/NCSOW.jar
```

Figure 7-97 shows the classpath modifications for the Servlet Engine. Your entries will vary depending on the instance name.

5. Click **Apply**.

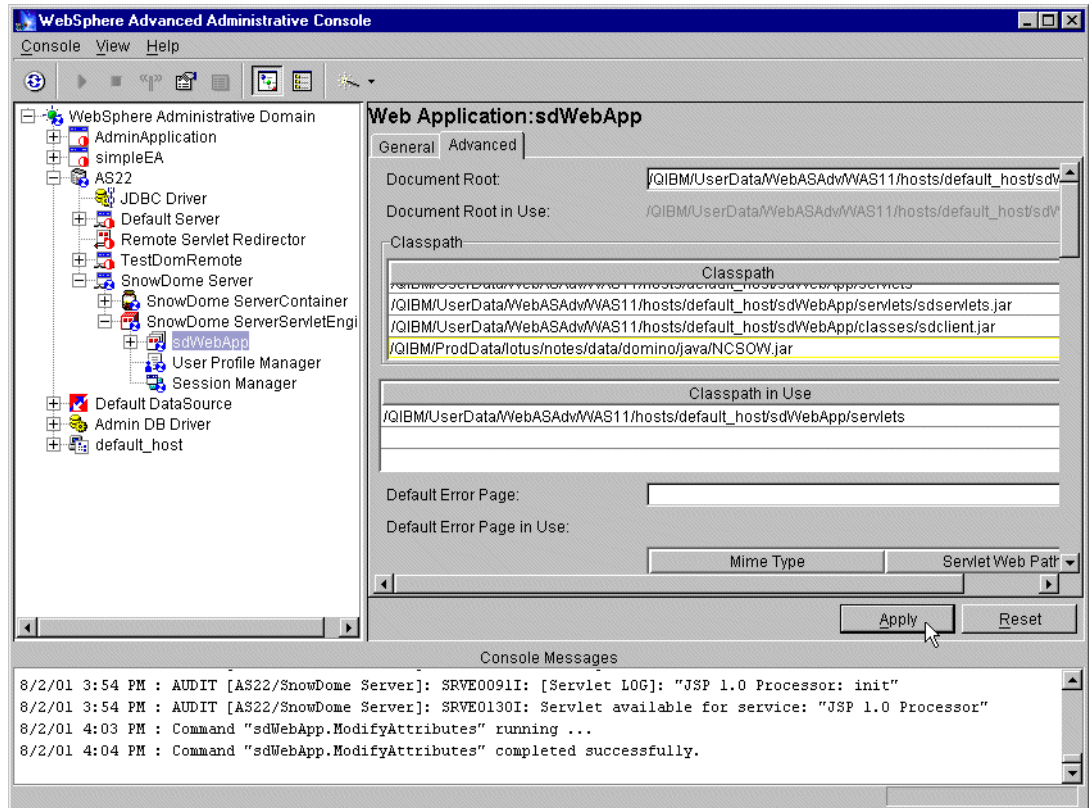


Figure 7-97 Modifying the servlet classpath

6. Restart the SnowDome server. Right-click **SnowDome Server** and click **Start**.
7. We are now ready to test the application. Start a Web browser and enter the following URL. You will need to modify the URL so that it points to your server.

`http://YOURSERVER.COM:8011/webapp/sdWebApp/SDIndex.html`

Figure 7-98 shows the Snow Dome Application entry window with the application running on the iSeries server.



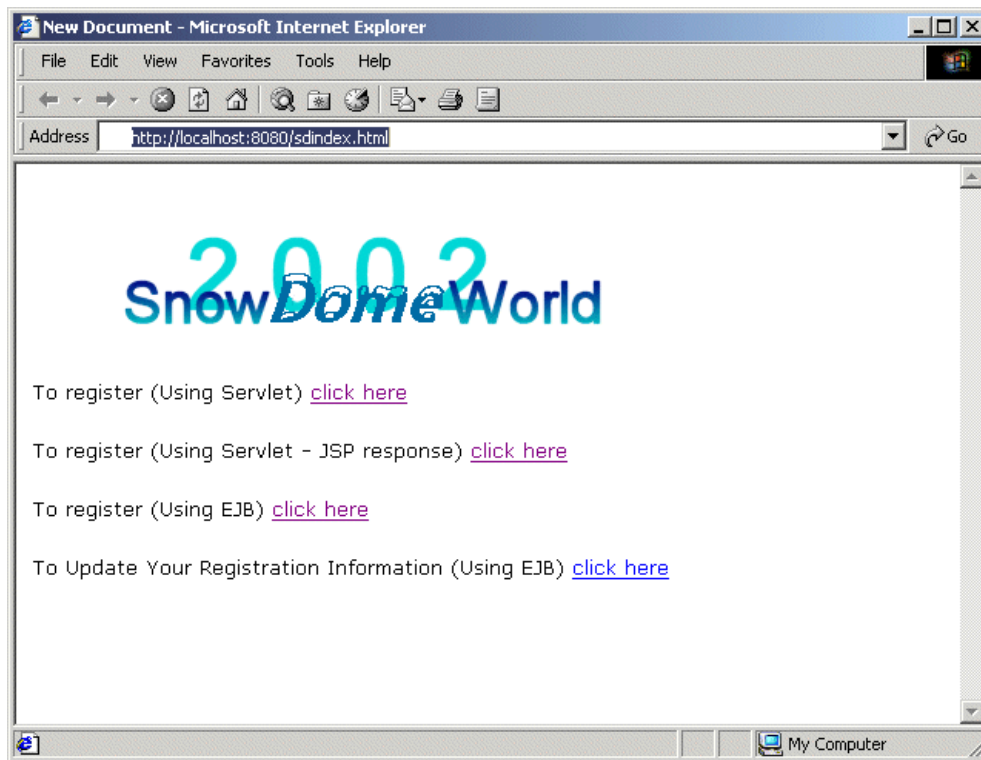


Figure 7-98 Entry window for the Snow Dome application

## 7.6 Using Enterprise JavaBeans from a Domino agent

This section covers the possible use of Domino Java agents to access EJBs managed in a WebSphere EJB container.

It is not possible to *directly* call EJBs from a Domino agent. The reason for this is that in Version 3.5.3 of the WebSphere Application Server, the Java Virtual Machine (JVM) is at level 1.2.2. Lotus Domino supports a JVM of 1.1.8. The two different JVMs cause errors in communication between the agent and the EJB. It is possible that future releases of Domino will include a JVM that is compatible with the JVM of your WebSphere instance. This would make calling EJBs possible, but there are still a number of concerns with this architecture:

- ▶ Domino manages threads in a way that is not optimal for EJB calls.
- ▶ WebSphere security cannot be enabled for an EJB called by a Domino agent. For most production environments, this is unacceptable.
- ▶ EJB classes need to be deployed to the file system. This does not conform with the way Domino agents are developed. Part of the attraction of agents is the ease of deployment via replication. The benefit is lost when the agent relies on files located on the file system.

There are two other methods available for calling EJBs from an agent. The first option is to call a servlet, and the servlet calls the EJB. The second method is to use a Remote Method Invocation (RMI) server to handle the call. An RMI server is a special kind of server that allows methods to be called remotely. Both these methods are possible and for an excellent example on using this technique, refer to *Domino and WebSphere Together Second Edition*, SG24-5955.



Before using agents to access EJBs, it is advisable to evaluate the requirements. Ask:

- ▶ Why does this call needs to be run from an agent?
- ▶ Is it possible to manage the call via a servlet?
- ▶ Can I achieve the same functionality without the use of an EJB?

In some circumstances, using EJBs from a Domino agent is a good solution. The flexibility of being able to schedule agents is an attraction to many developers. If you do decide to use EJBs and agents together, ensure that your development team and the administration staff understand the complexities.





## Part 3

# Appendixes

Part 3 of this redbook contains two appendixes. Appendix A, “Code examples” on page 265 contains the source code listings for the programs used in the Snow Dome example application covered in Chapter 7, “Accessing Domino from WebSphere” on page 183. Appendix B, “Additional material” on page 295 contains information on downloading and installing the actual material used for the Snow Dome example application.





# A

## Code examples

This appendix contains the source code for the examples used in Chapter 7, “Accessing Domino from WebSphere” on page 183 of this book.

## HTML and JSP files

This section contains the source code listings for the HTML and JSP files used in the Snow Dome example application used in Chapter 7, "Accessing Domino from WebSphere" on page 183.

### SDindex.html

```
<HTML>
<HEAD>
<TITLE> New Document </TITLE>
<META NAME="Generator" CONTENT="ITSO">
<META NAME="Author" CONTENT="Frodo Baggins">
<META NAME="Keywords" CONTENT="Snow Dome">
</HEAD>
<BODY>
<IMG src="snowdomebanner.gif" width="500" height="100" border="0" alt="Snow Dome World
2002">
<FONT SIZE="2" face="verdana,arial,geneva"><BR><BR>To register (Using Servlet) <A
HREF="registration.html">click here</A></FONT>
<FONT SIZE="2" face="verdana,arial,geneva"><BR><BR>To register (Using Servlet - JSP
response) <A HREF="regWJSP.html">click here</A></FONT>
<FONT SIZE="2" face="verdana,arial,geneva"><BR><BR>To register (Using EJB) <A
HREF="regejb.html">click here</A></FONT>
<FONT SIZE="2" face="verdana,arial,geneva"><BR><BR>To Update Your Registration Information
(Using EJB) <A HREF="SDsearch.jsp">click here</A></FONT>
</BODY>
</HTML>
```

### Registration.html

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <title>Snow Dome World Registration</title>
</head>
<body>
  &nbsp;
  <table COLS=2 WIDTH="56%" >
    <tr>
      <td><IMG src="snowdomebanner.gif" width="500" height="100" border="0" alt="Snow Dome World
2002"></td>
      <td></td>
    </tr>
  </table>
  <FONT SIZE="2" color="#FF3300" face="verdana,arial,geneva">Registration using servlet
only</FONT>
  <p><form method="post" action="servlet/snowdome.servlets.Register">
  <table BORDER=0 BGCOLOR=#E0E0E0 >
    <tr>
      <td>First Name:</td>
      <td><input type="text" name="firstName"></td>
    </tr>
    <tr>
      <td>Last Name:</td>
```

```

<td><input type="text" name="lastName"></td>
</tr>

<tr>
<td>E-Mail</td>

<td><input type="text" name="eMail"></td>
</tr>

<tr>
<td>Company</td>

<td><input type="text" name="company"></td>
</tr>

<tr>
<td>Phone</td>

<td><input type="text" name="phone"></td>
</tr>

<tr>
<td>Why Snow Domes are so cool</td>

<td><select name="whySnowDomes"><option>--Make a Selection--&nbsp;<option>Great
Souvenirs&nbsp;<option>They're pretty&nbsp;<option>Good collector's item&nbsp;<option>They
make nice gifts&nbsp;</select></td>
</tr>
</table>
<input type="submit" value="Submit"></form>
</body>
</html>

```

## RegWJSP.html

```

<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <meta name="GENERATOR" content="Mozilla/4.76 [en] (WinNT; U) [Netscape]">
  <title>Snow Dome World Registration</title>
</head>
<body>
&nbsp;
<table COLS=2 WIDTH="56%" >
<tr>
<td><IMG src="snowdomebanner.gif" width="500" height="100" border="0" alt="Snow Dome World
2002"></td>
<form method="post" action="servlet/snowdome.servlets.RegisterWJSP">
<td></td>
</tr>
</table>
<font SIZE="2" color="#FF3300" face="verdana,arial,geneva">Registration using servlet and
JSP response</font>
<p><table BORDER=0 BGCOLOR=#E0E0E0>
<tr>
<td>First Name:</td>

<td><input type="text" name="firstName"></td>
</tr>

```

```

<tr>
<td>Last Name:</td>

<td><input type="text" name="lastName"></td>
</tr>

<tr>
<td>E-Mail</td>

<td><input type="text" name="eMail"></td>
</tr>

<tr>
<td>Company</td>

<td><input type="text" name="company"></td>
</tr>

<tr>
<td>Phone</td>

<td><input type="text" name="phone"></td>
</tr>
<tr>
<td>Why Snow Domes are so cool</td>

<td><select name="whySnowDomes">
  <option>--Make a Selection--
  <option>Great Souvenirs
  <option>They're pretty
  <option>Good collector's item
  <option>They make nice gifts
</select>
</td>
</tr>

</table>
<input type="submit" value="Submit">
</form>
</body>
</html>

```

## Regejb.html

```

<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <meta name="GENERATOR" content="Mozilla/4.76 [en] (WinNT; U) [Netscape]">
  <title>Snow Dome World Registration</title>
</head>
<body>
  &nbsp;
  <table COLS=2 WIDTH="56%" >
  <tr>
  <td><IMG src="snowdomebanner.gif" width="500" height="100" border="0" alt="Snow Dome World
2002"></td>

  <td></td>
  </tr>
</table>

```



```

<font SIZE="2" color="#FF3300" face="verdana,arial,geneva">Registration using EJB</font>
<p><form method="post" action="servlet/snowdome.servlets.Controller">
<INPUT TYPE="HIDDEN" NAME="frmType" VALUE="regEJB">
<table BORDER=0 BGCOLOR=#E0E0E0 >
<tr>
<td>First Name:</FONT></td>

<td><input type="text" name="firstName"></td>
</tr>

<tr>
<td>Last Name:</td>

<td><input type="text" name="lastName"></td>
</tr>

<tr>
<td>E-Mail</td>

<td><input type="text" name="eMail"></td>
</tr>

<tr>
<td>Company</td>

<td><input type="text" name="company"></td>
</tr>

<tr>
<td>Phone</td>

<td><input type="text" name="phone"></td>
</tr>

<tr>
<td>Why Snow Domes are so cool</td>

<td><select name="whySnowDomes"><option>--Make a Selection--&nbsp;<option>Great
Souvenirs&nbsp;<option>They're pretty&nbsp;<option>Good collector's item&nbsp;<option>They
make nice gifts&nbsp;</select></td>
</tr>
</table>
<input type="submit" value="Submit"></form>
</body>
</html>

```

## SDSearch.jsp

```

<%@ page language="java" %>
<%@ page import="java.util.*" %>

<html>
<head><title>Snow Dome World 2002 - Registration Update</title></head>
<body>
<IMG src="snowdomebanner.gif" width="500" height="100" border="0" alt="Snow Dome World
2002">
<BR>
<FONT SIZE="2" face="verdana,arial,geneva" color="#FF3300">Search for registration details
using eMail</FONT>
<BR>

```

```

<BR>
<BR>

<FORM METHOD="post" ACTION="/webapp/sdWebApp/servlet/snowdome.servlets.Controller">
<INPUT TYPE="HIDDEN" NAME="frmType" VALUE="profileFind">

<%
    String errorMsg = (String)request.getAttribute("errMsg");
    if (errorMsg != null) {
        out.println("<FONT COLOR=#CC0000>Unable to find your registration information. Please
try again? <BR></FONT>");
    }
%>
<BR>
<FONT SIZE="2" face="verdana,arial,geneva">Your Email Address:</FONT><INPUT TYPE="text"
NAME="eMail" ID="eMail" SIZE="20" MAXLENGTH="20" >
<INPUT VALUE="Find" TYPE="submit">
</FORM>

</body>
</html>

```

## SDattende.jsp

```

<%@ page language="java" %>
<%@ page import="java.io.*" %>
<%@ page import="java.util.*" %>
<%@ page import="java.text.*" %>
<%@ page import="snowdome.ejb.*" %>

<%

RegDocAccessBean regab = (RegDocAccessBean)request.getAttribute("regDocBean");
String firstname = regab.getFirstName();
String lastname = regab.getLastName();
String email = regab.getEmail();
String company = regab.getCompany();
String phone = regab.getPhone();
String why = regab.getWhy();
String regNo = regab.getRegNumber();

%>

<html>
<head><title>Snow Dome World 2002 - Registration Details</title></head>
<body>
<IMG src="/webapp/sdWebApp/snowdomebanner.gif" width="500" height="100" border="0"
alt="Snow Dome World 2002">

<BR>
<FORM METHOD="post" ACTION="snowdome.servlets.Controller">
<INPUT TYPE="HIDDEN" NAME="frmType" VALUE="profileUpdate">
<INPUT TYPE="HIDDEN" NAME="regNo" VALUE=<%= regNo %>>
<H2><FONT SIZE="3" face="verdana,arial,geneva" COLOR="#000099">Registration information for
<%= firstname %></FONT></h2>
<BR>
<TABLE BORDER=0 BGCOLOR=#E0E0E0>
<TR><TD>FirstName</TD>
<TD><INPUT TYPE="text" NAME="firstName" SIZE="20" MAXLENGTH="20" VALUE="<%= firstname
%>"></TD>

```

```

<TR><TD>LastName</TD>
<TD><INPUT TYPE="text" NAME="lastName" SIZE="20" MAXLENGTH="20" VALUE="<%= lastname
%>"></TD>
<TR><TD>Email</TD>
<TD><INPUT TYPE="text" NAME="eMail" SIZE="20" MAXLENGTH="20" VALUE="<%= email %>"></TD>
<TR><TD>Company Name</TD>
<TD><INPUT TYPE="text" NAME="company" SIZE="20" MAXLENGTH="20" VALUE="<%= company
%>"></TD>
<TR><TD>Phone Number</TD>
<TD><INPUT TYPE="text" NAME="phone" SIZE="20" MAXLENGTH="20" VALUE="<%= phone %>"></TD>
<tr>
<td>Why snow domes are so cool:</td>
<td>
<select name="whySnowDomes">
    <option><%= why %>
    <option>Great Souvenirs
    <option>They're pretty
    <option>Good collectors item
    <option>They make nice gifts
</select>
</td>
</TABLE>
<INPUT VALUE="Submit" TYPE="submit">
</FORM>

</body>
</html>

```

## SDsuccessUpdate.jsp

```

<%@ page language="java" %>
<%@ page import="java.io.*" %>
<%@ page import="java.util.*" %>
<%@ page import="java.text.*" %>
<%@ page import="snowdome.ejb.*" %>

<%

RegDocAccessBean regab = (RegDocAccessBean)request.getAttribute("regDocBean");
String firstname = regab.getFirstName();
String lastname = regab.getLastName();
String companyname = regab.getCompany();
String phonenumber = regab.getPhone();
String why = regab.getWhy();
String email = regab.getEmail();

%>

<html>
<head><title>Successful Update</title></head>
<body>
<IMG src="/webapp/sdWebApp/snowdomebanner.gif" width="500" height="100" border="0"
alt="Snow Dome World 2002">
<BR>

<H2><FONT SIZE="3" COLOR="#000099" face="verdana,arial,geneva">Thanks <%= firstname %>. You
details have been updated as follows:</FONT></H2><BR>
<TABLE BORDER=0 BGCOLOR=#E0E0E0>

```

```

<TR>
  <TD>First Name</TD>
  <TD><%= firstname %></TD>
</TR>
<TR>
  <TD>Last Name</TD>
  <TD><%= lastname %></TD>
</TR>
<TR>
  <TD>Email</TD>
  <TD><%= email %></TD>
</TR>

<TR>
  <TD>Company</TD>
  <TD><%= companyname %></TD>
</TR>
<TR>
  <TD>Phone</TD>
  <TD><%= phonenumber %></TD>
</TR>
<TR>
  <TD>Why Snow Domes are so cool: </TD>
  <TD><%= why %></TD>
</TR>
</TABLE>
<BR>
Return to <A HREF="/webapp/sdWebApp/SDIndex.html">home page</A>..
</body>
</html>

```

## CustDisplay.jsp

```

<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <meta name="GENERATOR" content="Mozilla/4.76 [en] (WinNT; U) [Netscape]">
  <title>Customer Record</title>
</head>
<body>
<b><font color="#000099"><font size=+2>Customer Profile</font></font></b><jsp:useBean
id="CustDisplay" class="domwasis.CustInfoBean" scope="request" />
<br><b>Domino Data displayed using JSP and JavaBean</b>
<br>&nbsp;
<br>&nbsp;
<table BORDER =
<tr>
<td><b>Name:</b></td>

<td><jsp:getProperty name="CustDisplay" property="custName" /></td>
</tr>

<tr>
<td><b>Phone:</b></td>

<td><jsp:getProperty name="CustDisplay" property="custPhone" /></td>
</tr>

<tr>
<td><b>Company:</b></td>

```

```

<td><jsp:getProperty name="CustDisplay" property="custCompany" /></td>
</tr>

<tr>
<td><b>Email:</b></td>

<td><jsp:getProperty name="CustDisplay" property="custEmail" /></td>
</tr>
</table>

<br>&nbsp;
</body>
</html>

```

## ReturnMessage.jsp

```

<HTML>
<HEAD>
<%@page language="java" %>
<META name="GENERATOR" content="IBM WebSphere Page Designer V3.5 for Windows">
<META http-equiv="Content-Style-Type" content="text/css">
<TITLE>
Registration Confirmation
</TITLE>
</HEAD>

<BODY BGCOLOR="#FFFFFF">
<TABLE border="0">
  <TBODY>
    <TR>
      <TD><IMG src="/snowdomebanner.gif" width="500" height="100" border="0"></TD>
      <TD></TD>
    </TR>
  </TBODY>
</TABLE>
<P align="left"><FONT color="#000099" size="5" face="Arial">Registration Complete
</FONT><BR>
<BR>
<FONT face="Arial">Thank you <%=request.getParameter("firstName")%>
<%=request.getParameter("lastName")%> for registering for Snow Dome World 2001<BR>
<BR>
Your registration was complete on <%= new java.util.Date() %>.

<BR><BR>Return to <A HREF="/webapp/sdWebApp/SDIndex.html">home page</A>
</BODY>
</HTML>

```

## GetCustomer.html

```

<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <meta name="GENERATOR" content="Mozilla/4.76 [en] (WinNT; U) [Netscape]">
  <title>Customer Record</title>
</head>
<body>
<b><font color="#000099"><font size=+2>Customer Profile
<form action="/servlet/domwasis.DisplayCustInfoServlet" method="post">

```

```

Search</font></font></b><b><font color="#000099"><font size=+2></font></font></b>
<p><b><font color="#000000"><font size=+1>Please Select a customer<select
name="custChoice">
<option>Alice Liddell</option>
<option>Jack Sprat</option>
<option> Peter Rabbit</option>
<option>Robinson Crusoe</option>
</select>
<input type="submit" value="search">
</font></font></b>
</form>
</body>
</html>

```

## Java servlets

This section contains the source code listings for the Java servlets used in the Snow Dome example application used in Chapter 7, “Accessing Domino from WebSphere” on page 183.

### SimpleRemote.java

```

package domwas1;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;

public class SimpleRemote extends HttpServlet {
    Session session;
    Database db;
    View view;
    Document doc;
    String dbName = "custtrack.nsf";
    String viewName="Customers";
    String serverName = "domwas11";
    String host = "domwas11:8011";
    String user = "Test User";
    String password = "domwas";
    String tableTop =
"<table><tr><th>Name</th><th>Telephone</th><th>Company</th><th>E-Mail</th></tr>";
    String tableBottom = "</table>";

    /**
     *This method is called when servlet is unloaded
     * recycle the session to reclaim memory
     * Creation date: (7/26/01 8:39:40 AM)
     */
    public void destroy() {
        try{
            if(this.session !=null)
                this.session.recycle();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

/**
 * Process incoming HTTP GET requests
 *
 * @param request Object that encapsulates the request to the servlet
 * @param response Object that encapsulates the response from the servlet
 */
public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    performTask(request, response);
}
/**
 * Initializes the servlet.
 */
public void init() {
    if (session == null){
        try{
            session = NotesFactory.createSession(host,user,password);

            //Set a reference to the Customer Tracking database
            db = session.getDatabase(serverName,dbName);

            //set a reference to the Customers view
            view = db.getView(viewName);

        }
        catch (NotesException e){
            System.out.println("Error creating Notes session");
            e.printStackTrace();
        }
    }
}

public void performTask(HttpServletRequest request, HttpServletResponse response) {
    try{
        response.setContentType("text/html");
        response.setHeader("Pragma", "No-cache");
        response.setDateHeader("Expires", 0);
        response.setHeader("Cache-Control", "no-cache");
        PrintWriter out = response.getWriter();
        StringBuffer sbuff = new StringBuffer();

        //get the first document in the view
        doc = view.getFirstDocument();

        //loop through all documents and extract values form each column
        while(doc != null){
            sbuff.append("<tr>");

            //Get the values in the document
            Vector colVector = doc.getColumnValues();
            Enumeration colValues = colVector.elements();

            //Loop through the vector containing the values from the document
            while(colValues.hasMoreElements()){
                Object fieldValue = colValues.nextElement();
                sbuff.append("<td>" + fieldValue.toString() + "</td>");
            }

            sbuff.append("</tr>");
            doc = view.getNextDocument(doc);
        }
    }
}

```

```

    }
    //Generate results page
    writeResult(out,tableTop + sbuff.toString() + tableBottom);
}
catch (Exception e){
    e.printStackTrace();
}
}
/**
 * Insert the method's description here.
 * Creation date: (7/21/01 10:31:23 AM)
 * @param out java.io.PrintWriter
 * @param text java.lang.String
 */
public void writeResult(PrintWriter out, String text) {
    out.println("<html><head><title>");
    out.println("Simple Remote Domino Servlet");
    out.println("</title></head><body>");
    out.println("<h1>" + "Customer Tracking - Remote Classes" + "</h1>");
    out.println(text);
    out.println("</body></html>");
    out.flush();
}
}

```

## SimpleLocal.java

```

package domwasis;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;
/**
 * This is a sample servlet that uses the local Domino classes to access a
 * Notes Database and display the contents of a view
 * Creation date: (7/11/01 1:35:01 PM)
 * @author: Administrator
 */
public class SimpleLocal extends HttpServlet {
    Session session;
    Database db;
    View view;
    Document doc;
    String tableTop =
"<table><tr><th>Name</th><th>Telephone</th><th>Company</th><th>E-Mail</th></tr>";
    String tableBottom = "</table>";
/**
 * Process incoming HTTP GET requests
 *
 * @param request Object that encapsulates the request to the servlet
 * @param response Object that encapsulates the response from the servlet
 */
public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    performTask(request, response);
}
/**
 * Process incoming requests for information

```



```

*
* @param request Object that encapsulates the request to the servlet
* @param response Object that encapsulates the response from the servlet
*/
public void performTask(HttpServletRequest request, HttpServletResponse response) {
    try{
        response.setContentType("text/html");
        response.setHeader("Pragma", "No-cache");
        response.setDateHeader("Expires", 0);
        response.setHeader("Cache-Control", "no-cache");
        PrintWriter out = response.getWriter();
        StringBuffer sbuff = new StringBuffer();

        //initialize the Notes Thread
        NotesThread.sinitThread();
        //obtain an instance of the session
        session=NotesFactory.createSession();

        //test to determine that Domino session was created
        if(session ==null){
            writeResult(out, "Domino session could not be established");
            return;
        }

        //Set a reference to the Customer Tracking database
        db = session.getDatabase("", "CustTrack.nsf");

        //test to make sure database exists
        if (!db.isOpen()) {
            writeResult(out, "Database not found");
        }

        else{
            //set a reference to the Customers view
            view = db.getView("Customers");

            //get the first document in the view
            doc = view.getFirstDocument();

            //loop through all documents and extract values from each column
            while(doc != null){
                sbuff.append("<tr>");

                //Get the values in the document
                Vector colVector = doc.getColumnValues();
                Enumeration colValues = colVector.elements();

                //Loop through the vector containing the values from the document
                while(colValues.hasMoreElements()){
                    Object fieldValue = colValues.nextElement();
                    sbuff.append("<td>" + fieldValue.toString() + "</td>");
                }

                sbuff.append("</tr>");
                doc = view.getNextDocument(doc);
            }

            //Generate results page
            writeResult(out, tableTop + sbuff.toString() + tableBottom);
        }
    }
}

```

```

    }

    }
    catch (Exception e){
        e.printStackTrace();
    }

    finally{
        // recycle the session and terminate the thread

        try {
            if (session !=null){
                session.recycle();
            }
        }
        catch (Exception e ){
            e.printStackTrace();
        }

        NotesThread.stermThread();
    }
}
/**
 * Insert the method's description here.
 * Creation date: (7/21/01 10:31:23 AM)
 * @param out java.io.PrintWriter
 * @param text java.lang.String
 */
public void writeResult(PrintWriter out, String text) {
    out.println("<html><head><title>");
    out.println("Simple Remote Domino Servlet");
    out.println("</title></head><body>");
    out.println("<h1>" + "Customer Tracking - Local Classes" + "</h1>");
    out.println(text);
    out.println("</body></html>");
    out.flush();
}
}

```

## Register.java

```

package snowdome.servlets;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;
/**
 * Insert the type's description here.
 * Creation date: (8/1/01 4:39:50 PM)
 * @author: Administrator
 */
public class Register extends HttpServlet {
    Session session = null;
    Database database = null;
    String host = "domwas11:8011";
    String serverName = "domwas11";
}

```

```

        String dbName = "registration.nsf";
        String ior;
    /**
     * Insert the method's description here.
     * Creation date: (8/2/01 8:30:29 AM)
     * @param request com.sun.server.http.HttpServiceRequest
     */
    public void createRegistration(HttpServletRequest request) {
        try {
            //Create a document
            Document doc = database.createDocument();
            //Populate the document with the information gathered from the form
            doc.replaceItemValue("form", "Registration");
            doc.replaceItemValue("FirstName", request.getParameter("firstName"));
            doc.replaceItemValue("LastName", request.getParameter("lastName"));
            doc.replaceItemValue("PhoneNumber", request.getParameter("phone"));
            doc.replaceItemValue("CompanyName", request.getParameter("company"));
            doc.replaceItemValue("Email", request.getParameter("eMail"));
            doc.replaceItemValue("WhySnowDomes", request.getParameter("whySnowDomes"));

            //Save the Document and Generate Registration ID
            doc.save(true);
        }

        catch (NotesException e) {
            e.printStackTrace();
        }
    }
    /**
     * Process incoming HTTP GET requests
     *
     * @param request Object that encapsulates the request to the servlet
     * @param response Object that encapsulates the response from the servlet
     */
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {

        performTask(request, response);

    }
    /**
     * Process incoming HTTP POST requests
     *
     * @param request Object that encapsulates the request to the servlet
     * @param response Object that encapsulates the response from the servlet
     */
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {

        performTask(request, response);

    }
    /**
     * Initializes the servlet.
     */
    public void init() {
        if (session == null){
            try{
                ior = NotesFactory.getIOR(host);
                session=NotesFactory.createSessionWithIOR(ior);
            }

```

```

        database = session.getDatabase(serverName, dbName);
    }
    catch (NotesException e){
        System.out.println("Error creating Notes session");
        e.printStackTrace();
    }
}
/**
 * Process incoming requests for information
 *
 * @param request Object that encapsulates the request to the servlet
 * @param response Object that encapsulates the response from the servlet
 */
public void performTask(HttpServletRequest request, HttpServletResponse response) {

    PrintWriter out = null;

    try {
        response.setContentType("text/html");
        out = response.getWriter();
        createRegistration(request);

    }
    catch (Exception e) {
        e.printStackTrace();
    }

    writeConfirmMsg(out, request);

}
/**
 * Insert the method's description here.
 * Creation date: (8/2/01 8:30:29 AM)
 * @param request com.sun.server.http.HttpServiceRequest
 */
public void writeConfirmMsg(PrintWriter out, HttpServletRequest request){
    String firstName = request.getParameter("firstName");
    String lastName = request.getParameter("lastName");
    //Generate a confirmation message to be displayed to the user
    out.println("<!DOCTYPE HTML PUBLIC \"-//W3C// DTD HTML 4.0 Transitional //EN\">");
    out.println("<html><head><title>Conference Registration
Complete</title></head><body>");
    out.println("<h1>Conference Registration Confirmation</h1><hr>");
    out.println("Thank-you, " + firstName + " " + lastName + " for registering for the
conference.<br>");
    out.println("We look forward to seeing you at Snow Dome World 2002");
    out.println("</body></html>");

}
}

```

## RegisterWJSP.java

```

package snowdome.servlets;

import java.io.*;

```

```

import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;
/**
 * Insert the type's description here.
 * Creation date: (8/1/01 4:39:50 PM)
 * @author: Administrator
 */
public class RegisterWJSP extends HttpServlet {
    Session session = null;
    Database database = null;
    String host = "domwas11:8011";
    String serverName = "domwas11";
    String dbName = "registration.nsf";
    String ior;
/**
 * Insert the method's description here.
 * Creation date: (8/2/01 8:30:29 AM)
 * @param request com.sun.server.http.HttpServiceRequest
 */
public void createRegistration(HttpServletRequest request) {
    try {
        //Create a document
        Document doc = database.createDocument();
        //Populate the document with the information gathered from the form
        doc.replaceItemValue("form", "Registration");
        doc.replaceItemValue("FirstName", request.getParameter("firstName"));
        doc.replaceItemValue("LastName", request.getParameter("lastName"));
        doc.replaceItemValue("PhoneNumber", request.getParameter("phone"));
        doc.replaceItemValue("CompanyName", request.getParameter("company"));
        doc.replaceItemValue("Email", request.getParameter("eMail"));
        doc.replaceItemValue("WhySnowDomes", request.getParameter("whySnowDomes"));

        //Save the Document and Generate Registration ID
        doc.save(true);
    }

    catch (NotesException e) {
        e.printStackTrace();
    }
}
/**
 * Process incoming HTTP GET requests
 *
 * @param request Object that encapsulates the request to the servlet
 * @param response Object that encapsulates the response from the servlet
 */
public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

    performTask(request, response);
}
/**
 * Process incoming HTTP POST requests
 *
 * @param request Object that encapsulates the request to the servlet
 * @param response Object that encapsulates the response from the servlet
 */

```

```

public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

    performTask(request, response);

}
/**
 * Initializes the servlet.
 */
public void init() {
    if (session == null){
        try{
            ior = NotesFactory.getIOR(host);
            session=NotesFactory.createSessionWithIOR(ior);
            database = session.getDatabase(serverName,dbName);

        }
        catch (NotesException e){
            System.out.println("Error creating Notes session");
            e.printStackTrace();
        }
    }
}
/**
 * Process incoming requests for information
 *
 * @param request Object that encapsulates the request to the servlet
 * @param response Object that encapsulates the response from the servlet
 */

public void performTask(HttpServletRequest request, HttpServletResponse response) {

    try {
        createRegistration(request);
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    sendConfirmMsg(request, response);

}

public void sendConfirmMsg(HttpServletRequest request, HttpServletResponse response){
    try{
        //Get the request dispatcher object
        String url="returnMessage.jsp";
        RequestDispatcher rd = getServletContext().getRequestDispatcher(url);

        //forward the request to returnMessage.jsp
        rd.forward(request,response);
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}
}

```

## Controller.jsp

```
package snowdome.servlets;

import javax.naming.*;
import java.rmi.*;
import javax.ejb.*;
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
import java.util.*;
import snowdome.ejb.*;
/**
 * Insert the type's description here.
 * Creation date: (7/13/01 9:03:08 AM)
 * @author: Administrator
 */
public class Controller extends javax.servlet.http.HttpServlet {
/**
 * Process incoming HTTP GET requests
 *
 * @param request Object that encapsulates the request to the servlet
 * @param response Object that encapsulates the response from the servlet
 */
public void doGet(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response) throws javax.servlet.ServletException,
java.io.IOException {

    performTask(request, response);

}
/**
 * Process incoming HTTP POST requests
 *
 * @param request Object that encapsulates the request to the servlet
 * @param response Object that encapsulates the response from the servlet
 */
public void doPost(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response) throws javax.servlet.ServletException,
java.io.IOException {

    performTask(request, response);

}
/**
 * Initializes the servlet.
 */
public void init() {
    // insert code to initialize the servlet here
}
/**
 * Process incoming requests for information
 *
 * @param request Object that encapsulates the request to the servlet
 * @param response Object that encapsulates the response from the servlet
 */
public void performTask(HttpServletRequest request, HttpServletResponse response) {

    ServletContext sc = getServletContext();
```

```

RequestDispatcher rd = null;
String jspFile = null;

try {
    jspFile = process(request, response);
    rd = sc.getRequestDispatcher(jspFile);
    rd.forward(request, response);
}

catch (ObjectNotFoundException obje) {
    jspFile = "/SDsearch.jsp";
    try {
        request.setAttribute("errMsg", "true");
        rd = sc.getRequestDispatcher(jspFile);
        rd.forward(request, response);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

catch (RemoteException remoteException) {
    System.out.println("Session maybe invalid. Lets try again");
    try {
        jspFile = process(request, response);
        rd = sc.getRequestDispatcher(jspFile);
        rd.forward(request, response);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

catch (Throwable theException) {
    System.out.println("Are you sure you have IIOP running and accessible?");
    theException.printStackTrace();
}

}
/**
 * Insert the method's description here.
 * Creation date: (7/20/01 11:34:00 AM)
 * @return java.lang.String
 */
private String process(HttpServletRequest request, HttpServletResponse response)
    throws RemoteException, FinderException, NamingException, CreateException {

    String jspFile = null;

    //Lets determine which form called the servlet
    //If the request is to locate a valid profile do this..
    if (request.getParameter("frmType").equals("profileFind")) {
        RegDocAccessBean regab = new RegDocAccessBean();
        regab = regab.findByEmail(request.getParameter("eMail"));
        request.setAttribute("regDocBean", regab);
        jspFile = "SDAttendee.jsp";
    }
    //If the request is to update a reg document, do this..
    if (request.getParameter("frmType").equals("profileUpdate")) {
        jspFile = "SDsuccessUpdate.jsp";
    }
}

```



```

        RegDocKey key = new RegDocKey(request.getParameter("regNo"));
        RegDocAccessBean regab = new RegDocAccessBean(key);
        regab.setFirstName(request.getParameter("firstName"));
        regab.setLastName(request.getParameter("lastName"));
        regab.setCompany(request.getParameter("company"));
        regab.setEMail(request.getParameter("eMail"));
        regab.setPhone(request.getParameter("phone"));
        regab.setWhy(request.getParameter("whySnowDomes"));
        request.setAttribute("regDocBean", regab);
        regab.commitCopyHelper();
    }
    //It the request is to register, do this..
    if (request.getParameter("frmType").equals("regEJB")) {
        jspFile = "SDsuccessUpdate.jsp";
        RegDocAccessBean regab = new RegDocAccessBean();
        regab.setFirstName(request.getParameter("firstName"));
        regab.setLastName(request.getParameter("lastName"));
        regab.setCompany(request.getParameter("company"));
        regab.setEMail(request.getParameter("eMail"));
        regab.setPhone(request.getParameter("phone"));
        regab.setWhy(request.getParameter("whySnowDomes"));
        request.setAttribute("regDocBean", regab);
        regab.commitCopyHelper();
    }

    return jspFile;
}
}

```

## JavaBean

This section contains the source code listing for the JavaBean used in the Snow Dome example application used in Chapter 7, “Accessing Domino from WebSphere” on page 183.

### CustInfoBean.java

```

package domwasis;
public class CustInfoBean {
    private String custName;
    private String custEmail;
    private String custCompany;
    private String custPhone;
    public CustInfoBean(String custName, String custPhone, String custCompany, String
custEmail) {
        super();
        this.custName= custName;
        this.custPhone= custPhone;
        this.custCompany = custCompany;
        this.custEmail = custEmail;
    }
    public String getCustCompany() {
        return custCompany;
    }
    public String getCustEmail() {
        return custEmail;
    }
    public String getCustName() {

```

```

        return custName;
    }
    public String getCustPhone() {
        return custPhone;
    }
}

```

## Enterprise JavaBean

This section contains the source code listings for the Enterprise JavaBean used in the Snow Dome example application used in Chapter 7, “Accessing Domino from WebSphere” on page 183.

### RegDocBean.java

```

package snowdome.ejb;

import java.rmi.RemoteException;
import java.security.Identity;
import java.util.Properties;
import javax.ejb.*;
import lotus.domino.*;

/**
 * This is an Entity Bean class with BMP fields
 */
public class RegDocBean implements EntityBean {
    private javax.ejb.EntityContext entityContext = null;
    private final static long serialVersionUID = 3206093459760846163L;

    public java.lang.String firstName;
    public java.lang.String lastName;
    public java.lang.String eMail;
    public java.lang.String phone;
    public java.lang.String company;
    public java.lang.String why;
    public java.lang.String regNumber;

    private static Session session = null;
    private static Database database = null;
    private static View view = null;
    private Document doc = null;

    /**
     * This method was generated by the VisualAge for Java EJB AccessBean tool.
     */
    /* WARNING: THIS METHOD WILL BE REGENERATED. */
    public java.util.Hashtable _copyFromEJB() {
        com.ibm.ivj.ejb.runtime.AccessBeanHashtable h = new
        com.ibm.ivj.ejb.runtime.AccessBeanHashtable();

        h.put("regNumber", getRegNumber());
        h.put("firstName", getFirstName());
        h.put("company", getCompany());
        h.put("EMail", getEMail());
        h.put("why", getWhy());
        h.put("phone", getPhone());
        h.put("lastName", getLastName());
    }
}

```

```

        h.put("__Key", getEntityContext().getPrimaryKey());

        return h;
    }
    /**
     * This method was generated by the VisualAge for Java EJB AccessBean tool.
     */
    /* WARNING: THIS METHOD WILL BE REGENERATED. */
    public void _copyToEJB(java.util.Hashtable h) {
        java.lang.String localFirstName = (java.lang.String) h.get("firstName");
        java.lang.String localCompany = (java.lang.String) h.get("company");
        java.lang.String localEMail = (java.lang.String) h.get("EMail");
        java.lang.String localWhy = (java.lang.String) h.get("why");
        java.lang.String localPhone = (java.lang.String) h.get("phone");
        java.lang.String localLastName = (java.lang.String) h.get("lastName");

        if ( h.containsKey("firstName") )
            setFirstName((localFirstName));
        if ( h.containsKey("company") )
            setCompany((localCompany));
        if ( h.containsKey("EMail") )
            setEMail((localEMail));
        if ( h.containsKey("why") )
            setWhy((localWhy));
        if ( h.containsKey("phone") )
            setPhone((localPhone));
        if ( h.containsKey("lastName") )
            setLastName((localLastName));

    }
    /**
     * This method manages the domino server connectivity and creates the working document
     * Creation date: (7/16/01 4:40:05 PM)
     * @param argCaller java.lang.String
     * @param argMail java.lang.String
     * @param primaryKey smowdome.ejb.RedDocKey
     */
    private Document createSession(String argCaller, String argEmail, RegDocKey primaryKey) {
        //We first test to see if a session obj has been created.
        if (session == null) {
            System.out.println("No Current Session. Lets create one. Caller is ->" +
argCaller);
            try {
                String ior = NotesFactory.getIOR("domwas11:8011");
                session = NotesFactory.createSessionWithIOR(ior);
                database = session.getDatabase("domwas11", "registration.nsf");
                view = database.getView("Attendees");
            }

            catch (NotesException e) {
                System.out.println("Error creating session.");
                e.printStackTrace();
            }
        }

        //Now we have a session we need to test if it is valid
        //If the session is invalid we re-set it to null.
        //The caller of the bean (the servlet) can now recall the bean method
        try {

```

```

        if (session != null) {
            if (argCaller == "ejbCreate" ) {
                doc = database.createDocument();
            }
            if (argCaller == "FindByEmail") {
                doc = view.getDocumentByKey(argEmail, true);
            }
            if (argCaller == "FindByPrimaryKey") {
                doc = database.getDocumentByID(primaryKey.primaryKey);
            }
        }
    }
    catch (Exception e) {
        e.printStackTrace();
        session = null;
        database = null;
        view = null;
        doc = null;
    }

    return doc;
}

/**
 * ejbActivate method comment
 * @exception java.rmi.RemoteException The exception description.
 */
public void ejbActivate() throws java.rmi.RemoteException {}

/**
 * ejbCreate method for for the RegDoc bean
 * @return snowdome.ejb.RegDocKey
 * @param key argEmail
 * @exception javax.ejb.CreateException The exception description.
 * @exception java.rmi.RemoteException The exception description.
 */
public RegDocKey ejbCreate(String argEmail) throws CreateException, RemoteException {

    //Inititalize all fields to null except email which is arg;
    eMail = argEmail;
    firstName = null;
    lastName = null;
    company = null;
    phone = null;
    why = null;
    regNumber = null;

    try {
        doc = createSession("ejbCreate", null, null);
        doc.save();
        regNumber = doc.getNoteID();
    }
    catch (NotesException e) {
        System.out.println("Exception while creating document");
        e.printStackTrace();
    }
    RegDocKey key = new RegDocKey(regNumber);
    return key;
}

```

```

/**
 * This method returns a document from domino based on the email address passed in.
 * @return snowdome.ejb.RegDocKey
 * @param java.lang.String argEmail
 * @exception java.rmi.RemoteException The exception description.
 * @exception javax.ejb.FinderException The exception description.
 */
public snowdome.ejb.RegDocKey ejbFindByEmail(java.lang.String argEmail)
    throws java.rmi.RemoteException, javax.ejb.FinderException {

    try {
        doc = createSession("FindByEmail", argEmail, null);
        // The Object Not Found Exception is caught by the servlet and handled
        if (doc == null)
            throw new ObjectNotFoundException("Unable to find Registration Document");
        regNumber = doc.getNoteID();
    }

    catch (NotesException e) {
        e.printStackTrace();
    }

    RegDocKey primaryKey = new RegDocKey(regNumber);
    return primaryKey;
}
/**
 * This method returns a document from domino based on the NoteID passed in via as the
 * primaryKey
 * @return snowdome.ejb.RegDocKey
 * @param primaryKey snowdome.ejb.RegDocKey
 * @exception java.rmi.RemoteException The exception description.
 * @exception javax.ejb.FinderException The exception description.
 */
public snowdome.ejb.RegDocKey ejbFindByPrimaryKey(snowdome.ejb.RegDocKey primaryKey)
    throws java.rmi.RemoteException, javax.ejb.FinderException {
    doc = createSession("FindByPrimaryKey", null, primaryKey);

    // The Object Not Found Exception is caught by the servlet.
    if (doc == null)
        throw new ObjectNotFoundException("Unable to find Registration Document");
    return primaryKey;
}
/**
 * ejbLoad provides consistency of data between the domino database and the bean
 * @exception java.rmi.RemoteException The exception description.
 */
public void ejbLoad() throws java.rmi.RemoteException {
    try {
        //Included a call to createSession for debugging purposes
        RegDocKey key = (RegDocKey) getEntityContext().getPrimaryKey();
        doc = database.getDocumentByID(key.primaryKey);
        firstName = doc.getItemValueString("FirstName");
        lastName = doc.getItemValueString("LastName");
        company = doc.getItemValueString("CompanyName");
        phone = doc.getItemValueString("PhoneNumber");
        eMail = doc.getItemValueString("Email");
        why = doc.getItemValueString("WhySnowDomes");
        regNumber = key.primaryKey;
    }
    catch (NotesException e) {

```

```

        System.out.println("Error in ejbLoad method");
        e.printStackTrace();
    }
}
/**
 * ejbPassivate method comment
 * @exception java.rmi.RemoteException The exception description.
 */
public void ejbPassivate() throws java.rmi.RemoteException {}
/**
 * ejbPostCreate method for a BMP entity bean
 * @param key snowdome.ejb.RegDocKey
 * @exception java.rmi.RemoteException The exception description.
 */
public void ejbPostCreate(snowdome.ejb.RegDocKey key) throws java.rmi.RemoteException {}
/**
 * ejbRemove method comment
 * @exception java.rmi.RemoteException The exception description.
 * @exception javax.ejb.RemoveException The exception description.
 */
public void ejbRemove() throws java.rmi.RemoteException, javax.ejb.RemoveException {}
/**
 * ejbStore updates the domino document with changes
 * @exception java.rmi.RemoteException The exception description.
 */
public void ejbStore() throws java.rmi.RemoteException {
    // This method is called to update the data in domino with the beans data.
    try {
        //Call to create session included for debugging
        RegDocKey key = (RegDocKey) getEntityContext().getPrimaryKey();
        doc = database.getDocumentByID(key.primaryKey);
        doc.replaceItemValue("Form", "Registration");
        doc.replaceItemValue("FirstName", firstName);
        doc.replaceItemValue("LastName", lastName);
        doc.replaceItemValue("Email", eMail);
        doc.replaceItemValue("CompanyName", company);
        doc.replaceItemValue("PhoneNumber", phone);
        doc.replaceItemValue("WhySnowDomes", why);
        doc.save();
    }
    catch (NotesException e) {
        System.out.println("Error in ejbStore method");
        e.printStackTrace();
    }
}
/**
 * Insert the method's description here.
 * Creation date: (7/16/01 3:04:23 PM)
 * @return java.lang.String
 */
public java.lang.String getCompany() {
    return company;
}
/**
 * Insert the method's description here.
 * Creation date: (7/16/01 3:03:33 PM)
 * @return java.lang.String
 */
public java.lang.String getEmail() {
    return eMail;
}

```

```

    }
    /**
     * getEntityContext method comment
     * @return javax.ejb.EntityContext
     */
    public javax.ejb.EntityContext getEntityContext() {
        return entityContext;
    }
    /**
     * Insert the method's description here.
     * Creation date: (7/16/01 2:56:30 PM)
     * @return java.lang.String
     */
    public java.lang.String getFirstName() {
        return firstName;
    }
    /**
     * Insert the method's description here.
     * Creation date: (7/16/01 3:03:21 PM)
     * @return java.lang.String
     */
    public java.lang.String getLastName() {
        return lastName;
    }
    /**
     * Insert the method's description here.
     * Creation date: (7/16/01 3:04:05 PM)
     * @return java.lang.String
     */
    public java.lang.String getPhone() {
        return phone;
    }
    /**
     * Insert the method's description here.
     * Creation date: (7/16/01 3:05:41 PM)
     * @return java.lang.String
     */
    public java.lang.String getRegNumber() {
        return regNumber;
    }
    /**
     * Insert the method's description here.
     * Creation date: (7/16/01 3:04:46 PM)
     * @return java.lang.String
     */
    public java.lang.String getWhy() {
        return why;
    }
    /**
     * Insert the method's description here.
     * Creation date: (7/16/01 3:04:23 PM)
     * @param newCompany java.lang.String
     */
    public void setCompany(java.lang.String newCompany) {
        company = newCompany;
    }
    /**
     * Insert the method's description here.
     * Creation date: (7/16/01 3:03:33 PM)
     * @param newEMail java.lang.String

```

```

    */
    public void setEmail(java.lang.String newEMail) {
        eMail = newEMail;
    }
    /**
     * setEntityContext method comment
     * @param ctx javax.ejb.EntityContext
     * @exception java.rmi.RemoteException The exception description.
     */
    public void setEntityContext(javax.ejb.EntityContext ctx) throws java.rmi.RemoteException {
        entityContext = ctx;
    }
    /**
     * Insert the method's description here.
     * Creation date: (7/16/01 2:56:30 PM)
     * @param newFirstName java.lang.String
     */
    public void setFirstName(java.lang.String newFirstName) {
        firstName = newFirstName;
    }
    /**
     * Insert the method's description here.
     * Creation date: (7/16/01 3:03:21 PM)
     * @param newLastName java.lang.String
     */
    public void setLastName(java.lang.String newLastName) {
        lastName = newLastName;
    }
    /**
     * Insert the method's description here.
     * Creation date: (7/16/01 3:04:05 PM)
     * @param newPhone java.lang.String
     */
    public void setPhone(java.lang.String newPhone) {
        phone = newPhone;
    }
    /**
     * Insert the method's description here.
     * Creation date: (7/16/01 3:05:41 PM)
     * @param newRegNumber java.lang.String
     */
    private void setRegNumber(java.lang.String newRegNumber) {
        regNumber = newRegNumber;
    }
    /**
     * Insert the method's description here.
     * Creation date: (7/16/01 3:04:46 PM)
     * @param newWhy java.lang.String
     */
    public void setWhy(java.lang.String newWhy) {
        why = newWhy;
    }
    /**
     * unsetEntityContext method comment
     * @exception java.rmi.RemoteException The exception description.
     */
    public void unsetEntityContext() throws java.rmi.RemoteException {
        entityContext = null;
    }
}

```



## VisualAge for Java configuration file

This section contains the VisualAge for Java configuration file used in the Snow Dome example application used in Chapter 7, “Accessing Domino from WebSphere” on page 183.

### default\_app.webapp

```
<?xml version="1.0"?>
<webapp>
  <name>default</name>
  <description>default application</description>
  <error-page>/ErrorReporter</error-page>

  <servlet>
    <name>ErrorReporter</name>
    <description>Default error reporter servlet</description>
    <code>com.ibm.servlet.engine.webapp.DefaultErrorReporter</code>
    <servlet-path>/ErrorReporter</servlet-path>
    <autostart>true</autostart>
  </servlet>

  <servlet>
    <name>invoker</name>
    <description>Auto-registration servlet</description>
    <code>com.ibm.servlet.engine.webapp.InvokerServlet</code>
    <servlet-path>/servlet/*</servlet-path>
    <autostart>true</autostart>
  </servlet>

  <servlet>
    <name>jsp</name>
    <description>JSP support servlet</description>

    <!--
      ***
      *** Replace the JSP compiler with the required specification level.
      ***

      *** JSP 0.91 Compiler ***
      <code>com.ibm.ivj.jsp.debugger.pagecompile.IBMPageCompileServlet</code>

      *** JSP 1.0 Compiler ***
      <code>com.ibm.ivj.jsp.runtime.JspDebugServlet</code>

      *** JSP 1.1 Compiler ***
      <code>com.ibm.ivj.jsp.jasper.runtime.JspDebugServlet</code>
    -->
    <code>com.ibm.ivj.jsp.runtime.JspDebugServlet</code>

    <init-parameter>
      <name>workingDir</name>
      <value>${server_root$/temp/default_app</value>
    </init-parameter>
    <init-parameter>
      <name>jspemEnabled</name>
      <value>true</value>
    </init-parameter>
    <init-parameter>
      <name>scratchdir</name>
```

```

        <value>${server_root$/temp/JSP1_0/default_app</value>
    </init-parameter>
    <init-parameter>
        <name>keepgenerated</name>
        <value>true</value>
    </init-parameter>
    <autostart>true</autostart>
    <servlet-path>*.jsp</servlet-path>
</servlet>

<servlet>
    <name>file</name>
    <description>File serving servlet</description>
    <code>com.ibm.servlet.engine.webapp.SimpleFileServlet</code>
    <servlet-path></servlet-path>
    <init-parameter>
        <name></name>
        <value></value>
    </init-parameter>
    <autostart>true</autostart>
</servlet>
<servlet>
    <name>controller</name>
    <description>Controller Servlet for Snow Dome</description>
    <code>snowdome.servlets.Controller</code>
    <servlet-path>/webapp/sdWebApp/servlet/snowdome.servlets.Controller</servlet-path>
    <init-parameter>
        <name></name>
        <value></value>
    </init-parameter>
    <autostart>false</autostart>
</servlet>
<servlet>
    <name>register</name>
    <description>Servlet used for registration</description>
    <code>snowdome.servlets.Register</code>
    <servlet-path>/webapp/sdWebApp/servlet/snowdome.servlets.Register</servlet-path>
    <init-parameter>
        <name></name>
        <value></value>
    </init-parameter>
    <autostart>false</autostart>
</servlet>
<servlet>
    <name>registerwjsp</name>
    <description>Servlet used with JSP confirmation</description>
    <code>snowdome.servlets.RegisterWJSP</code>
    <servlet-path>/webapp/sdWebApp/servlet/snowdome.servlets.RegisterWJSP</servlet-path>
    <init-parameter>
        <name></name>
        <value></value>
    </init-parameter>
    <autostart>false</autostart>
</servlet>
</webapp>

```



## Additional material

This redbook refers to additional material. It can be downloaded from the Internet and installed into VisualAge for Java as described below.

### Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG246223>

Alternatively, you can go to the IBM Redbooks Web site at:

[ibm.com/redbooks](http://ibm.com/redbooks)

Click **Redbooks Online** and then **Additional materials**. Open the directory that corresponds with the redbook form number, SG246223.

### Using the Web material

The additional Web material that accompanies this redbook includes the following:

<i>File name</i>	<i>Description</i>
<b>SG246223.zip</b>	This file contains the following three ZIP files:
<b>NotesDB.zip</b>	This file contains two Domino databases. These are used in the examples in Chapter 6, "Using WebSphere and the Domino object models" on page 151, and Chapter 7, "Accessing Domino from WebSphere" on page 183.
<b>repository.zip</b>	This file contains the VisualAge for Java repository file. It includes all Java source for Chapter 6, "Using WebSphere and the Domino object models" on page 151, and Chapter 7, "Accessing Domino from WebSphere" on page 183.

**JSPandHTML.zip** This file contains the JSPs, HTML and image files used in Chapter 6, “Using WebSphere and the Domino object models” on page 151, and Chapter 7, “Accessing Domino from WebSphere” on page 183.

## Installing the Web material

The included material is intended for use with VisualAge for Java Enterprise Edition 3.5.3 and Lotus Domino R5.0.8.

### **SG246223.zip**

This archive contains the following files:

- ▶ NotesDB.zip
- ▶ repository.zip
- ▶ JSPandHTML.zip

Extract these files to a temporary directory. Details on installing and using these files are included in the following sections.

### **NotesDB.zip**

The NotesDB.zip file contains the files shown in Table B-1.

*Table B-1 Files in NotesDB.zip*

File	Role
CustTrack.nsf	This database is used for the examples in Chapter 6, “Using WebSphere and the Domino object models” on page 151.
registration.nsf	This database is used in Chapter 7, “Accessing Domino from WebSphere” on page 183.

1. Unzip the contents of NotesDB.zip into the data directory of your Domino iSeries server.
2. Change the ownership of the two files to QNOTES. Using the OS/400 command line, issue the following two commands. Make sure you replace the Domino server name with your instance name.

```
CHGOWN OBJ('/DOMINO/DOMWAS11/registration.nsf') NEWOWN(QNOTES)
CHGOWN OBJ('/DOMINO/DOMWAS11/CustTrack.nsf') NEWOWN(QNOTES)
```

3. Test the databases by opening them with the Domino client.
4. Using the Domino Administration client, sign the databases using an appropriate Notes ID.

### **JSPandHTML.zip**

The JSPandHTML file contains the files shown in Table B-2.

*Table B-2 Files in JSPandHTML.zip*

File name	Role
CustDisplay.jsp	This file is used for the examples in Chapter 6, “Using WebSphere and the Domino object models” on page 151.
GetCustomer.jsp	This file is used for the examples in Chapter 6, “Using WebSphere and the Domino object models” on page 151.

File name	Role
regejb.html	This file is used in Section 7.5.2, "Using the WebSphere Test Environment" on page 241. Refer to this section for details on its use.
registration.html	This file is used in Section 7.2.1, "Servlet access to Domino" on page 197. Refer to this section for details on its use.
regWJSP.html	This file is used in Section 7.3.1, "Extending the Snow Dome application using JSPs" on page 211. Refer to this section for details on its use.
returnMessage.jsp	This file is used in Section 7.3.1, "Extending the Snow Dome application using JSPs" on page 211. Refer to this section for details on its use.
SDAttendee.jsp	This file is used in Section 7.5.2, "Using the WebSphere Test Environment" on page 241. Refer to this section for details.
SDIndex.html	This file is used in all sections of Chapter 7, "Accessing Domino from WebSphere" on page 183.
SDsearch.jsp	This file is used in Section 7.5.2, "Using the WebSphere Test Environment" on page 241. Refer to this section for details
SDsuccessupdate.jsp	This file is used in Section 7.5.2, "Using the WebSphere Test Environment" on page 241. Refer to this section for details
snowdomebanner.gif	This file is used in all sections of Chapter 7, "Accessing Domino from WebSphere" on page 183.

1. Unzip the contents of JSPandHTML.zip into the following directory:  
X:\VAJAVA\_INSTALL\_DIRECTORY\ide\project\_resources\IBM WebSphere Test environment\hosts\default\_host\default\_app\web
2. Refer to Section 7.5.2, "Using the WebSphere Test Environment" on page 241 for details on how these files are used in the application.

### ***Repository.zip***

The Repository.zip file contains one file, domwas.bat. This is a VisualAge for Java repository file. It contains the following two VisualAge projects:

- ▶ **Domino and WebSphere.** This contains the Java source for Chapter 6, "Using WebSphere and the Domino object models" on page 151.
- ▶ **Snow Dome Project.** This contains the source for Chapter 7, "Accessing Domino from WebSphere" on page 183.

To add these projects to your VisualAge for Java installation, follow these steps:

1. Extract the file to a temporary directory, for example X:\CODE\VAJAVA\.
2. Start VisualAge for Java and go to the Workbench.

3. From the menu, select **File -> Import**. This displays the Import SmartGuide as shown in Figure B-1. Select **Repository** and click **Next**.

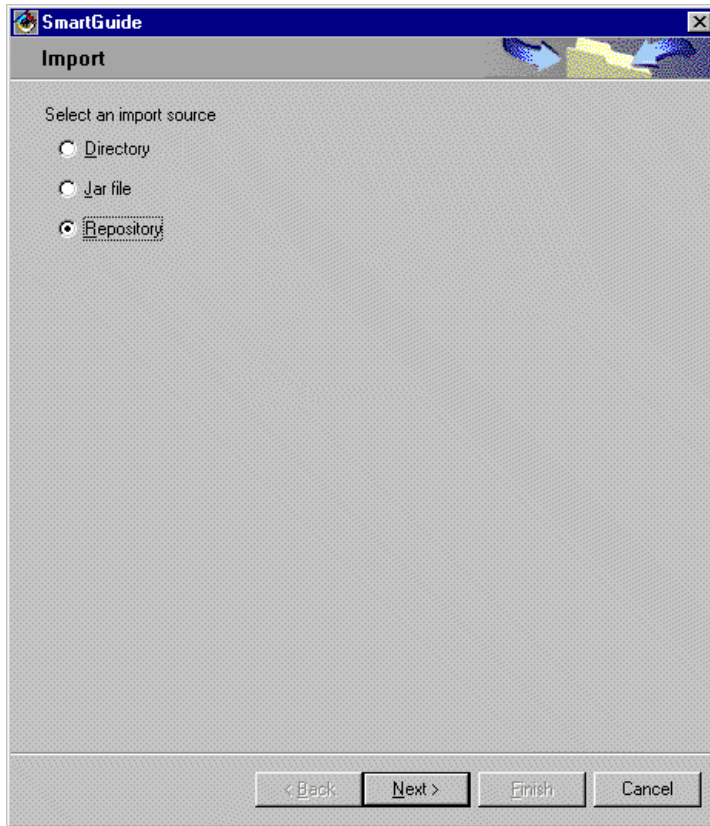


Figure B-1 Import SmartGuide

4. The Import from another repository window is displayed as shown in Figure B-2. From this window, specify the repository file located in the temporary directory.

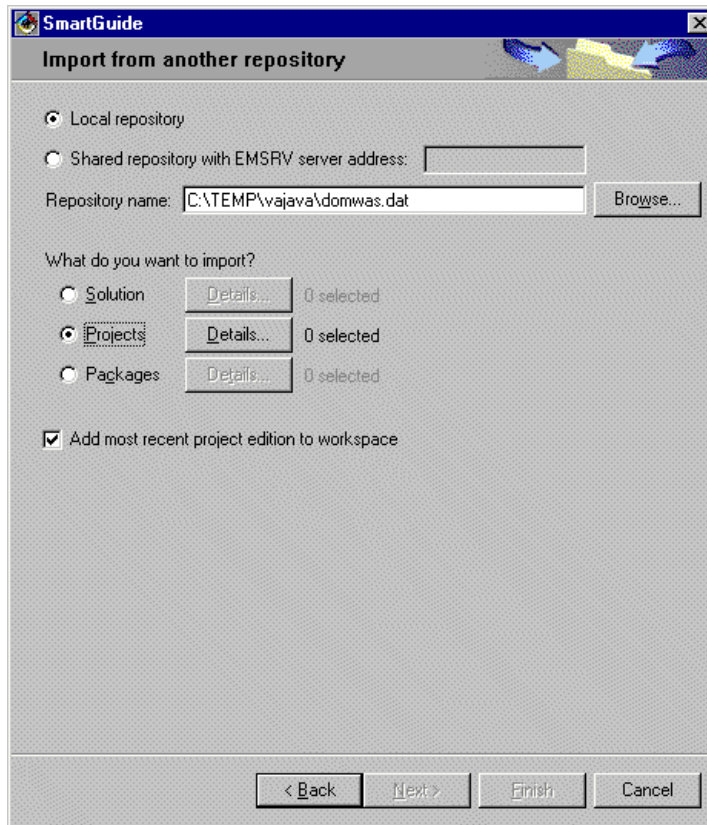


Figure B-2 Import from another repository

5. Select **Projects** as the source and click **Details**.
6. Figure B-3 shows the Project import window. Select the projects **Snow Dome Project** and **Domino and WebSphere**. The latest versions of the projects are automatically selected. Click **OK** to return to the Import from another repository window and then click **Finish**.

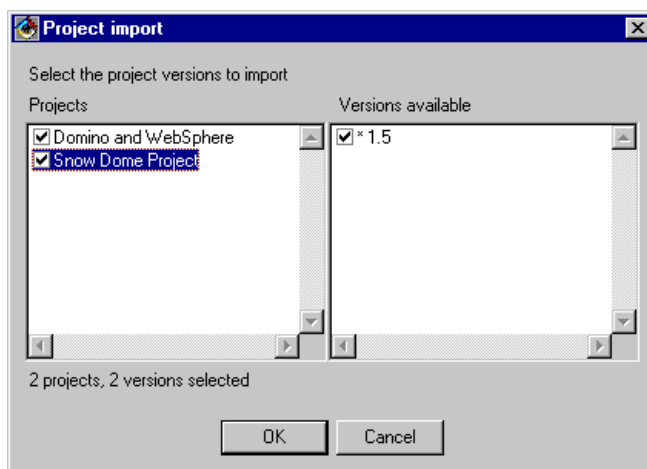


Figure B-3 Importing a project from a repository

7. VisualAge imports the two projects into your Workbench as shown in Figure B-4.

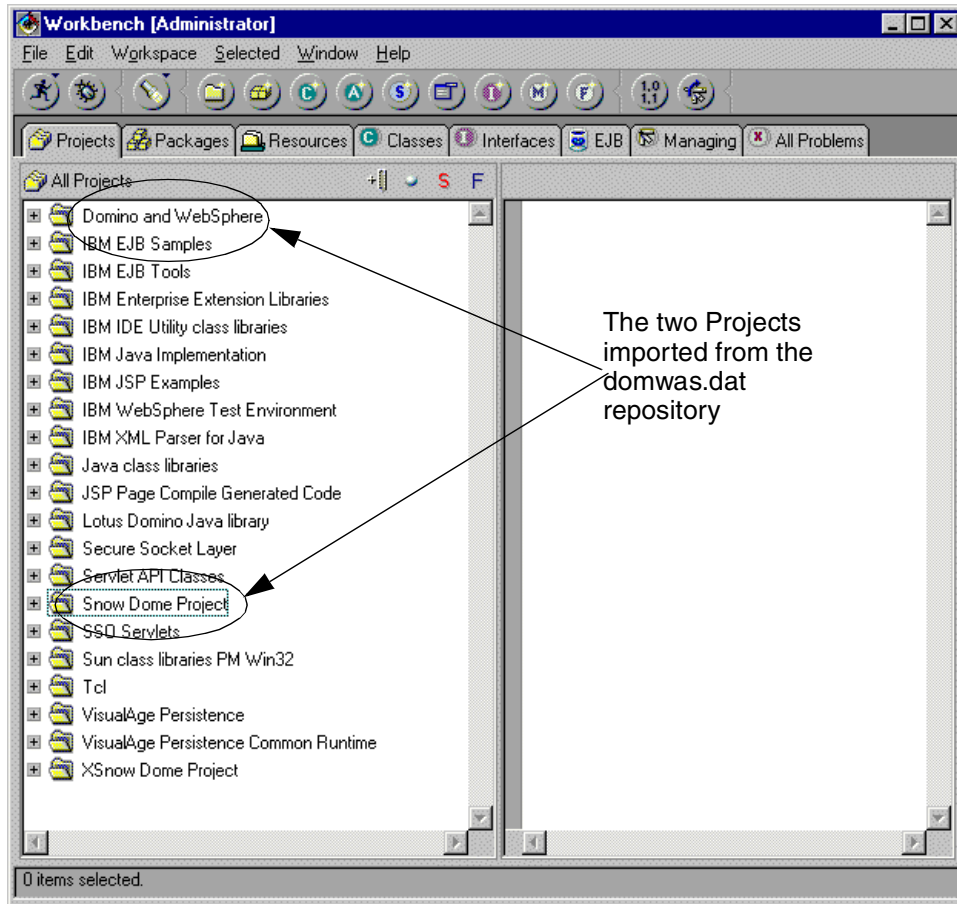


Figure B-4 VisualAge Workbench

8. This completes the installation of the Java source code.



# Special notices

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively

through The Open Group.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

## IBM Redbooks

For information on ordering these publications see “How to get IBM Redbooks” on page 304.

- ▶ *Lotus Domino for AS/400 R5: Implementation*, SG24-5592
- ▶ *Domino and WebSphere Together Second Edition*, SG24-5955
- ▶ *WebSphere V3.5 Handbook*, SG24-6161
- ▶ *Building iSeries Applications for WebSphere Advanced Edition 3.5*, SG24-5691
- ▶ *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, SG24-5755
- ▶ *EJB Development with VisualAge for Java for WebSphere Application Server*, SG24-6144

## Other resources

These publications are also relevant as further information sources:

- ▶ *Release Notes: Domino/Notes 5.05*, found at <http://www.notes.net> (click **Doc Library**)
- ▶ *HTTP Server for iSeries Webmaster's Guide*, GC41-5434

## Referenced Web sites

These Web sites are also relevant as further information sources:

- ▶ Domino for iSeries home page  
<http://www.ibm.com/servers/eserver/series/domino/>
- ▶ Lotus Developer Network home page  
<http://www.lotus.com/techzone/>
- ▶ iSeries Information Center  
<http://www.publib.boulder.ibm.com/pubs/html/as400/infocenter.htm>
- ▶ WebSphere Application Server for iSeries home page  
<http://www.ibm.com/servers/eserver/series/software/websphere/wsappserver/>
- ▶ WebSphere Application Server - Overview  
<http://www.ibm.com/software/webservers/appserv>
- ▶ IBM WebSphere InfoCenter - Online Documentation for WebSphere Application Server  
<http://www.ibm.com/software/webservers/appserv/infocenter.html>
- ▶ LDAP on the iSeries server  
<http://www.ibm.com/servers/eserver/series/ldap>

- ▶ Object Management Group Web site - source of information on CORBA  
<http://www.omg.org>
- ▶ *Lotus Domino and Notes R5 Best Practices Guide*  
<http://www.lotus.com/partners/bpg.nsf>
- ▶ Lotus Notes and Domino R5 Release Notes  
<http://www.notes.net/notesua.nsf/>

## How to get IBM Redbooks

Search for additional Redbooks or Redpieces, view, download, or order hardcopy from the Redbooks Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

Also download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

Redpieces are Redbooks in progress; not all Redbooks become Redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

## IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

# Glossary

**agent.** A Domino routine that automates tasks. Agents generally perform routine Domino tasks in the background but can also be started interactively by a user. You can program agents, using Java, LotusScript, or the Lotus Notes formula language.

**application programming interface (API).** A set of calling conventions defining how a service is invoked through a software package.

**authentication.** (1) Verification of the identity of a user or the user's eligibility to access an object. (2) Verification that a message has not been altered or corrupted. (3) A process used to verify the user of an information system or protected resources.

**authorization.** (1) The right granted to a user to communicate with or use a computer system. (2) An access right. (3) The process of granting a user either complete or restricted access to an object, resource, or function.

**certificate.** A message signed with a public key digital signature stating that a specified public key belongs to someone or something with a specified name.

**Certificate Authority.** An entity, typically a company, that issues digital certificates to other entities (organizations or individuals) to allow them to prove their identity to others.

**certification.** In Lotus Notes, the process of having an authorized (the certifier) person authenticate the identity of a user or server.

**client.** As in client/server computing, the application that makes requests to the server and often deals with the interaction necessary with the user.

**Common Object Request Broker Architecture (CORBA).** A language independent open standard for communication between object located across a network.

**database.** (1) A collection of interrelated data stored together with controlled redundancy according to a scheme to serve one or more applications. (2) All data files stored in the system. (3) A set of data stored together and managed by a database management system. (4) In Lotus Notes, a group of documents and their forms and views, stored under one file.

**DIOP.** The Domino Serve Task for IIOP.

**document.** In Domino, an object containing text, graphics, video, or audio data or any kind of rich text data.

**Domino.** Lotus Domino is an applications and messaging server with an integrated set of services that enable you to easily create secure, interactive business solutions for the Internet and corporate intranets.

**Extensible Markup Language (XML).** A standard metalanguage for defining markup languages that was derived from and is a subset of SGML. XML omits the more complex and less-used parts of SGML and makes it much easier to (a) write applications to handle document types, (b) author and manage structured information, and (c) transmit and share structured information across diverse computing systems. The use of XML does not require the robust applications and processing that is necessary for SGML. XML is being developed under the auspices of the World Wide Web Consortium (W3C).

**form.** In Lotus Notes, forms are used to author, read, and edit documents. Application developers design forms as templates for document creation, or as a way to view document data. Forms are comprised of other design elements such as text, fields, buttons, and programming code.

**host.** (1) In a computer network, a computer providing services such as computation, database access, and network control functions. (2) The primary or controlling computer in a multiple computer installation.

**Hypertext Markup Language (HTML).** A markup language that conforms to the SGML standard and was designed primarily to support the online display of textual and graphical information that includes hypertext links.

**Hypertext Transfer Protocol (HTTP).** In the Internet suite of protocols, the protocol that is used to transfer and display hypertext documents.

**Internet.** Large international, national, and regional backbone networks that allow local and campus networks and individuals access to global resources.

**Internet Inter-ORB Protocol (IIOP).** A specification for communication between Object Request Brokers (ORBS).

**intranet.** A TCP/IP network that is entirely under the control of a private authority or company. The intranet may have connections to other independent intranets (which would then be referred to as *extranets*) or the Internet. It may be fully or partially visible to the outside, depending on the implementation.

**Java archive (JAR).** A platform-independent file format that groups many files into one. JAR files are used for compression, reduced download time and security. Because the JAR format is written in Java, JAR files are fully extensible.

**JavaBeans.** A platform-independent, software component technology for building reusable Java components called “beans”. Once built, these beans can be made available for use by other software engineers or can be used in Java applications. Also, using JavaBeans, software engineers can manipulate and assemble beans in a graphical drag-and-drop development environment.

**Lightweight Directory Access Protocol (LDAP).** An open protocol that uses TCP/IP to provide access to directories that support an X.500 model and does not incur the resource requirements of the more complex X.500 Directory Access Protocol (DAP). Applications that use LDAP (known as directory-enabled applications) can use the directory as a common data store and for retrieving information about people or services, such as e-mail addresses, public keys, or service-specific configuration parameters.

**middleware.** A set of services that allow distributed applications to interoperate on a local area network or wide area network. It shields the developer or end user from the system complexity and enables the delivery of service requests or responses transparently across computing resources.

**NCSOW.jar.** A new JAR file called NCSOW (Notes Client-Side Objects for WebSphere) is provided in Domino R5.0.4 that allows an IBM WebSphere application to access the resources of a Domino server on a separate machine using the Domino back-end objects. Previously, WebSphere could only access Domino resources if they were running on the same machine.

**Object Request Broker (ORB).** A software component of CORBA that makes requests and returns responses on behalf of the object. Objects do not communicate with each other directly but rather they communicate through an ORB.

**Secure Sockets Layer (SSL).** A security protocol that provides communication privacy. SSL enables client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, and message forgery. SSL was developed by Netscape Communications Corp. and RSA Data Security, Inc.

**server.** Any computing resource dedicated to responding to client requests. Servers can be linked to clients through local area networks or wide area networks to perform services, such as printing, database access, fax, and image processing, on behalf of multiple clients at the same time.

**servlet.** A computer program that runs on a Web application server and responds to HTTP requests.

**Standard Generalized Markup Language (SGML).** A standard metalanguage for defining markup languages that is based on the ISO 8879 standard. SGML focuses on structuring information rather than presenting information. It separates the structure and content from the presentation. It also facilitates the interchange of documents across an electronic medium.

**transaction.** A unit of processing (consisting of one or more application programs) initiated by a single request. A transaction can require the initiation of one or more tasks for its execution.

**transaction processing.** A style of computing that supports interactive applications in which requests submitted by users are processed as soon as they are received. Results are returned to the requester in a relatively short period of time. A transaction processing system supervises the sharing of resources for processing multiple transactions at the same time.

**Transmission Control Protocol/Internet Protocol (TCP/IP).** A set of communication protocols that support peer-to-peer connectivity functions for both local and wide area networks.

**Uniform Resource Locator (URL).** The Internet address for a document, file, or other resource. It describes the protocol required to access the resource, the host where it can be found, and a path to the resource on that host.

**view.** (1) In Domino, views are used to sort, list, filter, and categorize documents. Application developers design views with particular selection criteria so that they will display lists of pertinent documents in a given order. (2) In a relational DBMS, a view is an efficient way of representing data without having to maintain it. It can include all or some of the columns or rows of the tables on which it is defined.

**Web.** See *World Wide Web*.

**Web browser.** The client component of the World Wide Web. The Web browser is responsible for formatting and displaying information, interacting with the user, and invoking external viewers for data types that it does not support directly. Examples of Web browsers are Netscape Communicator and Microsoft Internet Explorer.

**Web page.** Any document that can be accessed by a Uniform Resource Locator (URL) on the World Wide Web.

**Web server.** The server component of the World Wide Web. It is responsible for servicing requests for information from Web browsers. The information can be a file retrieved from the server's local disk or generated by a program called by the server to perform a specific application function.

**WebSphere.** A Web application server developed by IBM. It offers a standards-based Java server run-time environment and a set of Java development tools.

**workstation.** (1) A configuration of input and output equipment at which an operator works. (2) A terminal, a microcomputer or a network computer, usually one that is connected to a server, a mainframe or a network, at which a user can perform applications.

**World Wide Web.** (WWW) (W3) (the Web) An Internet client/server distributed information and retrieval system based on HTTP that transfers hypertext documents across a varied array of computer systems.

**XML.** See *Extensible Markup Language*.





# Index

## Symbols

\$WebSSOConfigs 145

## A

- access beans 234
- access control list (ACL) 123
  - updating with OS/400 profiles 93
- accessing Domino from WebSphere 183
- administration 15
- administrative workstation requirements 12
- agent 305
- alias 31, 45, 48, 167
  - for a Web site 31
  - multiple Domain Name System 46
- API 305
  - Domino Java 8, 153
  - LDAP C 58
  - OS/400 Directory Services 73
- application deployment 236
- application development environment 184
  - single sign-on 182
- application development programming 149
- application programming interface (API) 305
- application security 112
- application server 152, 251
- architecture 18
- attendees view 194
- authentication 7, 80, 102, 123, 305
  - challenge type 81, 102
  - fails accessing a protected resource 146
  - HTTP cookies 124
  - LTPA 80, 102
  - native OS 80, 102
  - using LDAP 78
  - WebSphere concepts 80, 102
- authorization 80, 102, 123, 305
  - fails accessing a protected resource 147
  - policy 109

## B

- B2B 2
- bean managed persistence (BMP) 218
  - creating 220
- BMP enterprise bean 220
- bootstrap.properties 22

## C

- callback methods 229
- calling WebSphere servlet 22
- certificate 305
- Certificate Authority 305
- certification 305

- CGI 3
- classpath 257
- client 305
- CMP 218
- code examples 14
- collaborative commerce 2
- common directory 7
- common errors 79
- Common Gateway Interface (CGI) 3
- common name 69
- Common Object Request Broker Architecture (CORBA) 154, 305
- configuring
  - Domino and WebSphere using Domino LDAP 101
  - Domino and WebSphere using OS/400 LDAP 80
  - Domino Directory for LDAP access 95
  - Domino HTTP server for WebSphere 20
  - Domino server document for SSO 133
  - Domino to use OS/400 LDAP 91
  - Domino URL mapping 33
  - Domino using iSeries HTTP server 35
  - Domino Web SSO configuration document 130
  - iSeries HTTP server instance 35
  - multiple DNS aliases 46
  - multiple virtual hosts with unique alias 48
  - multiple Web sites on Domino 27
  - OS/400 LDAP 61
  - SSO for Domino 130
  - SSO for WebSphere 125
  - virtual host in Domino 31
  - VisualAge for Java 186
  - WebSphere application security 112
  - WebSphere enterprise application 109
  - WebSphere resource security 115
  - WebSphere security permissions 118
  - WebSphere to access Domino objects 156
  - WebSphere to access Domino with local classes 169
  - WebSphere to access Domino with remote classes 156
  - WebSphere to use Domino LDAP 103
  - WebSphere to use OS/400 LDAP 81
- container managed persistence (CMP) 218
- Controller.jsp 283
- cookies 124, 136
- CORBA (Common Object Request Broker Architecture) 154, 305
- CustDisplay.jsp 272
- CustInfoBean.java 285
- custom finder method 233
- custom methods 226

## D

- da50.ntf 88
- database 305

- default server 24
- default virtual host 45
- default\_app 190
- default\_app.webapp 293
- delegation 80, 102
- deployment 242
- deployment code 234, 235
- DIOP 154, 305
- Directory Assistance database 88
- Directory Assistance Domino database 88
- directory entries 75
- Directory Information Tree (DIT) 58
- directory services 6
- directory sharing 59
  - using Domino LDAP 94
  - using LDAP 57
  - using OS/400 LDAP 60
- distinguished name (DN) 58, 68
- DIT (Directory Information Tree) 58
- DMT (Directory Management Tool) 75
- document 305
- Domino 305
  - access from WebSphere 183
  - access using local classes 169
  - accessing from EJBs 218
  - accessing from WebSphere 183
  - and WebSphere security without SSO enabled 137
  - calling EJBs from agent 260
  - compared with WebSphere 4
  - configuring CORBA 155
  - configuring Domino Directory for LDAP access 95
  - configuring Domino server document for SSO 133
  - configuring to use iSeries HTTP server 35
  - configuring to use OS/400 LDAP 91
  - different HTTP port 52
  - Directory Assistance database 88
  - directory sharing 59
  - DSAPI 21
  - enabling SSO 130
  - hosting multiple Web sites 27
  - importing LTPA keys 131
  - Java APIs 8
  - local objects 154
  - multi-server option 130
  - object models with WebSphere 151
  - plug-in for iSeries HTTP server 34
  - running with iSeries HTTP servers 52
  - server document 133
  - server fails to load the Web SSO Configuration document 145
  - servlets accessing 197
  - single sign-on 7
  - single sign-on prerequisites 124
  - switching from iSeries HTTP to Domino HTTP 52
  - updating database ACLs with OS/400 profiles 93
  - URL mapping 33
  - using OS/400 LDAP 60
  - verifying security with SSO 139
  - verifying security without SSO 137
  - verifying SSO between WebSphere and Domino 136
  - virtual host 26, 31
  - virtual host for WebSphere to recognize calls 53
  - Web SSO Configuration document 130
  - Web SSO Configuration document saving fails 145
  - WebSphere and OS/400 LDAP 80
- Domino database ACL 93
- Domino Directory for LDAP access 95
- Domino HTTP 25
  - call WebSphere servlet 22
  - local objects 154
  - serving WebSphere 7
  - single sign-on 135
- Domino HTTP server
  - support 19
  - to manage multiple Web sites 25
- Domino Internet Inter-ORB Protocol (DIOP) 154
- Domino Java API 8, 153
- Domino LDAP 94, 101
  - configuring Domino and WebSphere to use 101
  - configuring WebSphere to use 103
  - for WebSphere 101
  - starting 97
  - stopping 97
  - verifying connection 99
- Domino object model
  - local objects 8, 153, 154, 169
  - performance 182
  - remote objects 8, 153, 154
  - with WebSphere 151
- Domino plug-in for OS/400 HTTP server 7
- domino.srvpgm 21
- DSAPI 21

## E

- e-business strategy 2
- EJB (Enterprise JavaBeans) 217
- EJB container 152
- EJB Group 219
- EJB server 152
- EJB test client 236
- ejbCreate 230
- ejbFindByPrimaryKey 231, 233
- ejbLoad 231
- EJBs 4, 152
- ejbStore 232
- enabling
  - WebSphere global security 81, 103
  - WebSphere global security for SSO 125
- enterprise application 109, 121
  - starting 121
- Enterprise JavaBeans 4, 152, 217, 286
  - accessing Domino 218
  - adding custom methods 226
  - adding fields 223
  - callback methods 229
  - calling Domino agent 260
  - creating an EJB Group 219
  - creating BMP 220
  - EJB test client 236
  - ejbCreate 230

- ejbFindByPrimaryKey 231, 233
- ejbLoad 231
- ejbStore 232
- finder method 233
- groups 219
- remote interface 233
- entity bean 218
- environment 184
- ePerson object class 61, 68
- example of WebSphere accessing Domino 184
  - configuring VisualAge for Java 186
  - creating the Domino database 191
  - invoking servlets 197
  - site architecture 185
  - testing 206
  - using Enterprise JavaBeans 217
  - using JSPs 211
- examples 14
- exporting LTPA keys 129
- Extensible Markup Language (XML) 305, 307
- extranet 306

## F

- field 201
  - adding 223
- finder method 233
- firewall 156
- form 305

## G

- GetCustomer.html 273
- getSessionToken 182
- global security 81, 103
  - settings for WebSphere 125
  - updating for SSO 125

## H

- HMTL 305
- host 305
- hosting multiple Web sites 27
- HTML
  - and JSP files 266
- HTTP 6, 305
  - configuring Domino HTTP server for WebSphere 20
  - configuring Domino to use iSeries HTTP server 35
  - configuring iSeries HTTP server instance 35
  - Domino URL mapping 33
  - iSeries server 34
  - multiple iSeries HTTP instances 46
  - multiple Web sites 25, 44
  - running Domino and iSeries servers together 52
  - switching from iSeries HTTP to Domino HTTP 52
  - virtual host 45
- HTTP cookies 124, 136
- HTTP port 52
- HTTP serving options 17
- Hypertext Markup Language (HTML) 305
- Hypertext Transfer Protocol (HTTP) 305

## I

- IBM SecureWay Directory for OS/400 60
- IBM SecureWay Directory Management Tool (DMT) 75
- IBM WebSphere InfoCenter 159
- IFS 172
- IIOP 154, 305
- inetOrgPerson object class 73
- Integrated File System (IFS) 172
- integration 6
- Internet 305
- Internet Inter-ORB Protocol (IIOP) 154, 305
- Internet Service Provider (ISP) 45
- intranet 306
- invoking servlets 197
- iSeries
  - HTTP server to manage multiple Web sites 44
  - running with Domino HTTP servers 52
- iSeries server
  - configuring HTTP server instance 35
  - directory sharing 59
  - Domino LDAP 94
  - HTTP support 34
  - managing OS/400 LDAP directory 75
  - multiple HTTP instances 46
  - OS/400 LDAP 60
  - Qshell utility 74, 75, 101
  - requirements 12
  - switching from iSeries HTTP to Domino HTTP 52
  - virtual host 45
- ISP 45

## J

- JAR file 243, 305, 306
- Java APIs 8
- Java archive (JAR) 305, 306
- Java servlets 274
- Java to access Domino from WebSphere 153
- Java virtual machine (JVM) 159
- JavaBeans 285, 306
- JavaServer Pages (JSPs) 4, 152, 211
- JDBC 8
- JSP (JavaServer Pages) 4, 152, 211
- JSP and HTML files 266
- JVM 159

## L

- LDAP 6, 7, 58, 306
  - authentication 78
  - common name 69
  - configuring Domino Directory for access 95
  - directory 78
  - directory sharing 57
  - directory sharing on iSeries 59
  - distinguished name (DN) 68
  - Domino 94
  - ePerson object class 61, 68
  - inetOrgPerson object class 73
  - publisher object class 73
  - schema 61

- server job log 79
- which server to use? 125
- LDAP (Lightweight Directory Access Protocol) 58
- LDAP C API 58
- LDAP Data Interchange Format (LDIF) 77
- ldap\_bind 79
- LDIF (LDAP Data Interchange Format) 77
- libdomino.srvpgm 21
- Lightweight Directory Access Protocol (LDAP) 58, 306
- Lightweight Third Party Authentication (LTPA) 7, 80, 102, 127
  - exporting keys 129
  - importing keys into Domino 131
- local access versus remote access 182
- local classes 169
- local objects 153, 154
  - performance 182
- lotus.domino.Session 182
- LTPA 7, 80, 102, 127
- LTPA key 129

## M

- meta-directory 59
- methods 202
- middleware 2, 306
- multiple Domain Name System (DNS) aliases 46
- multiple iSeries HTTP instances 46
- multiple Web sites 25
  - hosting 27
  - iSeries server 44
  - managing with Domino HTTP server 25
  - managing with iSeries HTTP server 44

## N

- native OS authentication 80, 102
- NCSO.jar 153
- NCSOW.jar 153, 169, 306
- network computing framework 19
- new database 192
- node 152
- Notes.jar 153, 169
- NotesFactory method 154, 182

## O

- Object Management Group (OMG) 154
- object model 151
- Object Request Broker (ORB) 154, 306
- Operations Navigator 170, 171
- ORB 154, 306
- OS/400 Directory Services 60
  - APIs 73
- OS/400 HTTP server
  - Domino plug-in 7
- OS/400 LDAP 60, 74, 75, 78, 79
  - APIs 73
  - C APIs 77
  - configuring 61
  - configuring Domino and WebSphere to use 80

- configuring Domino to use 91
- configuring WebSphere to use 81
- managing directories 75
- preventing select SDD information from being published 73
- publishing SDD information 61, 68
- SDD to LDAP mapping 68
- starting and stopping 67
- verifying connection 74
- viewing the server job log 79
- WebSphere and Domino 80
- OS/400 prerequisites 61
- overview 1

## P

- packages 190
- permissions 124
- prerequisites 11
- problem determination 78, 145
  - Domino HTTP for single sign-on 135
  - OS/400 LDAP 79
- projects 189
- protected resource 146
- publisher object class 73
- publishing SDD users 69
- publishing with WebSphere Studio 245

## Q

- QDIRSRV job 79
- Qshell utility 74, 75, 101

## R

- Redbooks Web site 304
  - Contact us ix
- RegDocBean.java 286
- Regejb.html 268
- Register servlet 198
  - stub code 198
- Register.java 278
- RegisterWJSP servlet 212
- RegisterWJSP.java 280
- registration form 192
- Registration.html 266
- RegWJSP.html 267
- remote access versus local access 182
- remote classes 156
- remote interface 233
- remote objects 153, 154
  - performance 182
- requirements
  - iSeries server 12
  - workstation 12
- resource security 115
- restarting WebSphere Application Server 86
- ReturnMessage.jsp 273
- returnMessage.jsp 214

## S

- saving the LDAP directory 78
- schema 61
- scripting elements 211
- SDattendee.jsp 270
- SDD (System Distribution Directory) 68
- SDD users 69
  - preventing select users from being published 73
- SDindex.html 266
- SDSearch.jsp 269
- SDsuccessUpdate.jsp 271
- Secure Sockets Layer (SSL) 306
- SecureWay Directory 58
- securing WebSphere resources 109
- security
  - enabling WebSphere global security 81, 103
  - enabling WebSphere global security for SSO 125
  - firewall 156
  - verifying Domino and WebSphere SSO 137, 139
- security permission 118
- server 306
- server instance 13
- Servlet Engine 152
- servlets 3, 197, 306
  - accessing Domino 197
- session beans 218
- SGML 306
- SimpleLocal 276
- SimpleLocal servlet 169
- SimpleLocal.java 276
- SimpleRemote servlet 157
- SimpleRemote.java 274
- single sign-on (SSO) 6, 123, 124
  - across multiple Domino servers 7
  - application development 182
  - between Domino and WebSphere 7
  - configuring Domino server document for SSO 133
  - Domino Web SSO Configuration document 130
  - enabling Domino 130
  - enabling WebSphere 125
  - exporting LTPA keys from WebSphere 129
  - fails when accessing protected resources 147
  - importing LTPA keys into Domino 131
  - prerequisites 124
  - problem determination 145
  - verifying between WebSphere and Domino 136
- Snow Dome application 184
  - extending using JSPs 211
- software requirements 12
- source code 265
- SSL 306
- SSO 6, 124
- SSO (single sign-on) 123
- Standard Generalized Markup Language (SGML) 306
- starting
  - Domino LDAP 97
  - OS/400 LDAP 67
  - WebSphere enterprise application 121
- stateful 218
- stateless 218

stopping

- Domino LDAP 97

- OS/400 LDAP 67

supported features 7

System Distribution Directory (SDD)

- preventing select SDD information from being published 73

- publishing to LDAP 61, 68

- SDD to LDAP mapping 68

## T

TCP/IP 306

transaction 306

transaction processing 306

Transmission Control Protocol/Internet Protocol (TCP/IP) 306

troubleshooting single sign-on 145

## U

Uniform Resource Locator (URL) 306

URL 306

URL mapping 33

user registry 81, 103

using Qshell utility 74, 75, 101

## V

verifying

- connection to Domino LDAP 99

- connection to OS/400 LDAP 74

- Domino and WebSphere security with SSO 139

- Domino and WebSphere security without SSO 137

- SSO between WebSphere and Domino 136

view 306

virtual host 26, 27, 31, 45, 48

- for WebSphere to recognize Domino calls 53

- with unique alias 48

VisualAge for Java 152, 186

- adding required features 186

- configuration file 293

- creating a project 189

- creating packages 190

- modifying default\_app 190

## W

Web 306

Web application 25, 153

Web application classpath 257

Web browser 74, 100, 307

Web page 307

Web server 307

Web site alias 31

Web SSO Configuration document for Domino 130

- loading failure 145

- saving failure 145

WebSphere 2, 108, 307

- access to Domino 183

- accessing Domino 8, 183

- and Domino security without SSO enabled 137

- Application Server 152
    - architecture 18
    - authorization policy 109
    - bootstrap.properties 22
    - calling servlet with Domino HTTP 22
    - compared with Domino 4
    - configuring Domino HTTP server 20
    - configuring multiple DNS aliases 46
    - configuring to use Domino LDAP 101
    - creating an application server 251
    - default server 24
    - deploying applications 242
    - directory sharing 59
    - Domino and OS/400 LDAP 80
    - Domino object models 151
    - EJB container 152
    - EJB server 152
    - Enterprise JavaBeans 152
    - multiple virtual hosts with unique alias 48
    - node 152
    - resources 109
    - security 81, 103, 125
    - Servlet Engine 152
    - single sign-on 7
    - single sign-on prerequisites 124
    - using Domino HTTP 7
    - using Domino LDAP 94
    - using Java to access Domino 153
    - using OS/400 LDAP 60
    - verifying security with SSO 139
    - verifying security without SSO 137
    - verifying SSO between WebSphere and Domino 136
    - virtual host to recognize Domino calls 53
    - virtual hosts 27
    - Web application 153
  - WebSphere Application Server 3, 152, 156
    - accessing Domino objects 156
    - accessing Domino with local classes 169
    - accessing Domino with remote classes 156
    - authentication concepts 80, 102
    - configuring application security 112
    - configuring enterprise application 109
    - configuring resource security 115
    - configuring security permissions 118
    - configuring to use Domino LDAP 103
    - configuring to use OS/400 LDAP 81
    - creating an instance 13
    - default virtual host 45
    - deployment to 242
    - enabling global security 81, 103, 125
    - enabling SSO 125
    - exporting LTPA keys 129
    - global security 103
    - restarting 86
    - securing WebSphere resources 109
    - starting the enterprise application 121
    - topology 152
      - local objects example 169
      - remote objects 157
    - user registry 81, 103
    - Web application 25
    - WebSphere Studio 245
    - WebSphere Test Environment (WTE) 206, 241
    - workstation 307
    - workstation requirements 12
    - World Wide Web (WWW) 307
- X**
- X.500 standards 58
  - XML 305, 307
  - Xms parameter 159
  - Xmx parameter 159



Redbooks

## Domino and WebSphere Integration on the IBM @server iSeries Server

(0.5" spine)

0.475" <-> 0.873"

250 <-> 459 pages









**Redbooks**

# Domino and WebSphere Integration


on the IBM  iSeries Server

**Configure single sign-on between Domino and WebSphere on iSeries**

**Understand options for using LDAP and different HTTP stacks**

**Programmatically access Domino from WebSphere**

IBM WebSphere and Lotus Domino are both strategic products for e-business and business-to-business or B2B solutions. Domino and its related products deliver collaborative commerce services, while WebSphere allows for Java-based Web applications with a strong transactional emphasis.

This IBM Redbook documents the setup and configuration of an integrated environment with WebSphere and Domino on the IBM  iSeries server. Part 1 of this redbook focuses on administration topics, such as single sign-on (SSO) and the use of a common LDAP directory for authentication, as well as options for different HTTP stacks.

Part 2 of this redbook focuses on investigating the application development topic of how WebSphere can access Domino applications. It shows and discusses code snippets from example programs.

The examples in this redbook were developed using VisualAge for Java Enterprise Edition 3.5.3 and tested using WebSphere Application Server 3.5.3. To use this book effectively, you should be familiar with the Java programming language and the concepts of Enterprise JavaBeans.

**INTERNATIONAL  
TECHNICAL  
SUPPORT  
ORGANIZATION**

**BUILDING TECHNICAL  
INFORMATION BASED ON  
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
[ibm.com/redbooks](http://ibm.com/redbooks)

SG24-6223-00

ISBN 0738421936